MANAGING LIFETIME RELIABILITY, PERFORMANCE, AND POWER TRADEOFFS IN MULTICORE MICROARCHITECTURES

A Dissertation
Presented to
The Academic Faculty

By

William J. Song

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2015

Copyright © William J. Song 2015

MANAGING LIFETIME RELIABILITY, PERFORMANCE, AND POWER TRADEOFFS IN MULTICORE MICROARCHITECTURES

Approved by:

Dr. Sudhakar Yalamanchili, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Hyesoon Kim
School of Computer Science
Georgia Institute of Technology

Dr. Sung Kyu Lim
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Moinuddin Qureshi
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Saibal Mukhopadhyay
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Arun Rodrigues
Computer Science Research Institute
Sandia National Laboratories

Date Approved: 29 October 2015

ACKNOWLEDGEMENTS

This research was supported by Semiconductor Research Corporation (SRC) under tasks #2084.001 and #2318.001, Sandia National Laboratories, and GRC/SRC graduate fellowship from IBM.

I sincerely appreciate guidance and many constructive comments throughout Ph.D. studies from my advisor, Dr. Sudhakar Yalamanchili. It was a great honor to be his student at Georgia Tech. I also want to thank Dr. Saibal Mukhopadhyay, Dr. Yorai Wardi, and Dr. Hyesoon Kim for a series of collaborated research and publications as well as committee members, Dr. Sung Kyu Lim, Dr. Moinuddin Qureshi, and Dr. Arun Rodrigues.

I am also grateful for a number of technical research internship opportunities at

1) Sandia National Laboratories with Dr. Arun Rodrigues,

2) AMD Research with Dr. Gabe Loh,

3) IBM T.J. Watson Research Center with Dr. Pradip Bose and his team,

4) Qualcomm with Mr. Scott Runner and Mr. Jeff Gemar.

Lastly, I am really appreciative of all the supports from my family, friends, and CASL labmates.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Managing Lifetime Reliability, Performance, and Power Tradeoffs in Multicore Microarchitectures

William J. Song

115 Pages

Directed by Dr. Sudhakar Yalamanchili

The objective of this research is to characterize and manage lifetime reliability, microarchitectural performance, and power tradeoffs in multicore processors. This dissertation is comprised of three research themes; 1) modeling and simulation method of interacting multicore processor physics, 2) characterization and management of performance and lifetime reliability tradeoff, and 3) extending Amdahl's Law for understanding lifetime reliability, performance, and energy efficiency of heterogeneous processors. With continued technology scaling, processor operations are increasingly dominated by multiple distinct physical phenomena and their coupled interactions. Understanding these behaviors requires the modeling of complex physical interactions. This dissertation first presents a novel simulation framework that orchestrates interactions between multiple physical models and microarchitecture simulators to enable research explorations at the intersection of application, microarchitecture, energy, power, thermal, and reliability. Using this framework, workload-induced variation of device degradation is characterized, and its impacts on processor lifetime and performance are analyzed. This research introduces a new metric to quantify performance-reliability tradeoff. Lastly, the theoretical models of heterogeneous multicore processors are proposed for understanding performance, energy efficiency, and lifetime reliability consequences. It is shown that these system metrics are governed by Amdahl's Law and correlated as a function of processor composition, scheduling method, and Amdahl's scaling factor. This dissertation highlights the importance of multi-dimensional analysis and extends the scope of microarchitectural studies by incorporating the physical aspects of processor operations and designs.

CHAPTER I

INTRODUCTION

Microarchitectural operations and designs face physical challenges. Execution control schemes such as power or thermal management combined with inherent workload dynamics create spatiotemporal changes in thermal and voltage stresses. These in turn lead to variations in leakage power, logic delay, or device degradation across a multicore die, which impact system-level properties such as performance, power (or energy) efficiency, and lifetime reliability. *The analysis of future microarchitectures must be holistic including power, energy, thermal, and reliability concerns since their coupled interactions are becoming major determinants of processor performance*. This problem statement drives three research topics in this dissertation:

- Modeling and simulation method of interacting multicore processor physics
- Characterization and management of performance and lifetime reliability tradeoff
- Extending Amdahl's Law for understanding lifetime reliability, performance, and energy efficiency of heterogeneous processors

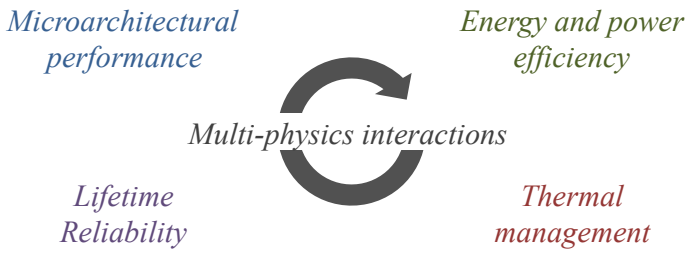


Figure 1: The analysis of future microarchitecture and applications must be *holistic* including power, energy, thermal, and reliability concerns since their coupled interactions are becoming major determinants of processor performance.

Microarchitecture cannot be solely evaluated by performance metrics. As power becomes a critical barrier to processors operations, the paradigm of designing processors has been shifting from simply improving performance to enhancing power (or energy) efficiency. Power efficiency can be naively increased by lowering operation voltage and clock frequency, but prolonged execution time may not meet a real-time deadline constraint or quality of service (QoS) requirement. This problem is referred to as *performance and power (or energy) efficiency tradeoff*.

High-performance operations generate more switching activities of microarchitecture components. These lead to increases in power and heat dissipations, which have an adverse impact on lifetime reliability. On the other hand, processor reliability enhancement favors low utilization to reduce stresses and resulting failures. Managing *performance and reliability tradeoff* is a challenging task in that reliability characteristics strongly depend on voltage and thermal stresses that are induced by microarchitectural operations and application behaviors.

As such, microarchitectural performance, power (or energy), and reliability are correlated in a complex loop as illustrated in Figure 1. These interactive phenomena in processors are referred to as *multi-physics interactions*. This dissertation fills the gap between microarchitecture and processor physics that have been treated as separate studies.

This dissertation is comprised of contents from relevant publications [58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69]. The following summarizes the major contributions of this dissertation.

1. **A method for architecture-level modeling and simulation of interacting multicore processor physics:**

   This dissertation addresses an important problem of modeling physical interactions in multicore processors. A novel, open-source framework named *KitFox* is introduced to realize a *multi-physics* simulation environment [58, 59, 63, 64, 66, 68, 69]. The

KitFox framework facilitates microarchitectural explorations at the intersection of application, microarchitecture, performance, and various physical phenomena including power, energy, thermal, and lifetime reliability. This framework is distinguished from conventional trace-driven or static simulation methods since multiple distinct physical models are incorporated and concurrently simulated in the same framework. KitFox internally handles interactions among various physical properties, thereby relieving microarchitecture modelers from orchestrating them. This research answers two important questions regarding multi-physics modeling, which have not been addressed in related efforts [5, 14, 15, 16, 25, 26, 50, 56, 87]; 1) error propagation through the chain of interacting physical models due to the inaccuracy of a model in the loop and 2) the simulation overhead of an integrated multi-physics simulation environment.

2. **Modularized integration of physical models and implementing standardized interface to microarchitecture simulators:**

   This work tackles a standardization problem when integrating various implementations of physical models into the same framework [63, 64, 66, 69]. A challenge is that developers write simulators and models in many different ways. KitFox implements a standard interface to bridge multiple physical models, where individual models are encapsulated into *libraries* and interchangeable. The notion of *pseudo components* and *hierarchy* enables users to flexibly compose processor description to simulate different package or microarchitecture designs. The pseudo components represent abstract physical units (e.g., packages, floorplans, circuit blocks) for which library models are attached to estimate physical properties. The pseudo component hierarchy serves to interconnect disparate modeling tools and transfer calculated results between the models. A set of application programming interface (API) functions is provided to ease the use of physical models with microarchitecture simulations. To the best of our knowledge, this is the first attempt to standardize the integration and interface of various physical modeling tools.

3. **Lifetime reliability modeling in microarchitecture simulations:**

   In dynamically controlled multicore processors, lifetime reliability cannot be analyzed in a static manner such as using an average or worst-case condition. This work incorporates lifetime reliability models to cycle-level microarchitecture simulations in conjunction with transient power and thermal modeling [60, 61, 62, 65, 67]. Reliability characteristics are subject to spatiotemporally varying voltage and thermal stresses. Multi-physics modeling implemented by KitFox enables the calculation of failure rates and resulting lifetime prediction with respect to varying stress conditions that are induced by workload dynamics (e.g., compute or memory-bound) or adaptive controls (e.g., turbo-mode execution). This is distinguished from prior work [14, 15, 16, 18, 27, 28, 34, 45, 46, 56, 85, 87] that relied on pre-generated traces or static methods (e.g., using average or representative power and thermal profiles), where the reliability impact of such dynamics could not be correctly captured.

4. **Characterization of workload-induced device degradation distribution on a multicore die and variance-aware reliability management:**

   This research characterizes workload-induced device degradation distribution in a multicore processor based on integrated microarchitecture and multi-physics simulations [62, 67]. Challenges are understanding 1) how the execution of parallel applications creates device degradation distribution on the multicore die and 2) how such degradation distribution affects system-level properties such as processor lifetime and performance. Results from integrated microarchitecture and multi-physics simulations reveal that the degradation distribution caused by parallel executions can be characterized by using a probabilistic model such as normal distribution. This observation leads to the fact that the variance of degradation distribution is a critical factor to determine processor lifetime. Therefore, controlling processor executions to reduce the variance of degradation distribution effectively leads to processor lifetime enhancement. This concept is referred to as *dynamic reliability variance management* [62, 67].

5. **Quantification of performance and lifetime reliability tradeoff in multicore processors and microarchitectural adaptations:**

   This research addresses an important fact that performance and lifetime reliability are inversely related [67]. A new metric, *throughput-lifetime product*, is introduced to quantify the tradeoff between performance and lifetime reliability. Results from integrated microarchitecture and multi-physics simulations show that this metric is maximized when both performance and reliability are balanced instead of performing throughput or reliability-oriented operations. In this study, three microarchitectural adaptation techniques are proposed to improve the performance-reliability tradeoff using the introduced throughput-lifetime product metric; 1) phase-aware thread migration, 2) variance-aware reliability management via voltage scaling and 3) turbo-mode execution with voltage scaling-based variance management. It is demonstrated via simulations that the proposed adaptation techniques can effectively improve throughput-lifetime product.

6. **Extending Amdahl's Law for modeling the performance, energy efficiency, and lifetime reliability of heterogeneous multicore processors:**

   Heterogeneous processors have been suggested as alternative microarchitectural designs to enhance performance and energy efficiency [12, 13, 17, 23, 30, 31, 38, 39, 47, 75, 83]. Using Amdahl's Law [4], heterogeneous models were primarily analyzed in performance and energy efficiency aspects to demonstrate their advantages over conventional homogeneous systems. This dissertation extends the heterogeneous multicore models presented in prior work [23, 83] for performance and energy calculations. This research proposes the lifetime reliability models of multicore processors. The proposed models show that the lifetime reliability, performance, and energy efficiency, of heterogeneous multicore processors are traded as a function of 1) core utilization (Amdahl's scaling factor), 2) processor composition (number of big and small cores), and 3) thread scheduling method. This research also presents a compact thermal estimation of hypothetical heterogeneous processors to accurately model the dependency between core

temperatures and device aging phenomena.

7. **Assessing the lifetime reliability, energy efficiency, and performance consequences of heterogeneous multicore processors:**

   This research evaluates the lifetime reliability, performance, and energy efficiency of heterogeneous processors using the proposed multicore models. It shows that a heterogeneous processor may have a serious reliability challenge. If the processor is comprised of only one big core and many small cores, stresses can be biased to the big core especially when workloads spend more time on sequential operations. This problem is identified as a *reliability bottleneck* in the heterogeneous processor. The analysis shows that incorporating a few big cores can mitigate the reliability bottleneck at the big cores and enhance processor lifetime with minor impact on performance and energy efficiency. However, adding too many big cores will have an adverse impact on lifetime reliability as well as performance since reduced parallel throughput increases total execution time and therefore failure rates.

This dissertation is organized as follows. Chapter 2 summarizes related work, encompassing three research topics of this dissertation. In Section 2.1, related efforts towards the development of integrated microarchitecture and multi-physics simulation environment are described, and distinctions of the proposed framework to these infrastructures are highlighted. Section 2.2 overviews general approaches to architecture-level lifetime reliability modeling and simulation, and previous research for dynamic reliability management is summarized. Section 2.3 discusses prior work for the performance and energy modeling of heterogeneous multicore processors.

Chapter 3 details the proposed multi-physics simulation framework, named KitFox. It first explains design motivation and summarizes the distinguished features of KitFox framework in Section 3.1. A concept of library integration of modeling tools is introduced,

and the modeling methods of individual library types are explained in Section 3.2. Section 3.3 illustrates the architecture-level abstraction of physical interactions in multicore processors. Section 3.4 presents a data manipulation method in the integrated simulation environment, and an error propagation study is discussed. Section 3.5 explains a processor description method via the notion of pseudo component hierarchy. Section 3.6 describes parallel simulation interfaces and implementations, and Section 3.7 evaluates the overhead of multi-physics simulations. Section 3.8 summarizes the applications of KitFox to a range of research problems that are not covered in this dissertation. These use cases demonstrate the versatility of KitFox framework. Finally, Section 3.9 lists possible improvements to KitFox for future work.

Chapter 4 presents the characterization and management of performance-reliability tradeoff in multicore processors. Section 4.1 explains an experiment method using KitFox for architecture-level reliability modeling. Section 4.2 presents the lifetime reliability characterization of a multicore processor caused by the execution of parallel applications. Then, the effects of degradation variation on processor lifetime are discussed. In Section 4.3, performance and lifetime reliability tradeoff is evaluated based on the simulation results, and a new metric is introduced to quantify and measure this tradeoff. Section 4.4 presents microarchitectural adaptations to enhance the performance-reliability tradeoff. Key insights obtained from the analysis are summarized in Section 4.5.

Chapter 5 discusses the lifetime reliability, performance, and energy efficiency consequences of heterogeneous multicore processors. In Section 5.1, performance and energy models of multicore processors presented in prior work are reviewed with updated assumptions. The implications of these models are discussed in Section 5.2. In Section 5.3, a method for compact thermal estimation is proposed to predict temperature differences between distinct multicore processors. Estimated temperature differences are applied to failure rate equations, and the lifetime reliability models of multicore processors are proposed

in Section 5.4. The multicore lifetime reliability models are evaluated in Section 5.5, particularly focusing on the reliability implications of heterogeneous processors. Section 5.6 applies the performance, energy, and lifetime reliability models to real benchmarks based on microarchitecture simulations. Section 5.7 ends the chapter and summarizes key insights learned from the analysis. Finally, the conclusion of this dissertation is presented.

CHAPTER II

RELATED WORK

With continued technology scaling and increased power and heat densities, processor operations and performance are increasingly dominated by physical problems. Microarchitectural approaches to mitigate these effects must be based on a profound understanding of how the processor physics is manifested in system-level properties such as microarchitectural performance, power (or energy) efficiency, and lifetime reliability. Prior work summarized in this chapter covers three related research topics of this dissertation; 1) modeling and simulation of multicore processor physics, 2) architecture-level lifetime reliability modeling and management of multicore processors, and 3) Amdahl's Law for performance and energy scaling of heterogeneous multicore processors.

2.1 Modeling and Simulation of Multicore Processor Physics

Modeling and simulation of future microarchitectures and applications require more than performance measurement and estimation since physical constraints and their coupled interactions have become major determinants of processor operations. The computer architecture community has invested considerable efforts to develop, validate, and release various physical modeling tools such as 1) power models of Wattach [11], Cacti [77] and its extended tool called McPAT [40], Orion [32] and its recent replacement named DSENT [76], DRAMsim [54], 2) thermal models of HotSpot [29], 3D-ICE [70], and a compact microfluidic cooling model in 3-dimensional integrated circuits (3D ICs) [80], and 3) lifetime reliability models such as bias temperature instability (BTI) presented in the work of Srinivasan et al. [71, 72, 74], White and Bernstein [82], and JEDEC standards [1]. The state of the practice is using these point tools in the independent steps of simulations such as a trace-drive simulation depicted in Figure 2(a). In the trace-driven simulation, a single

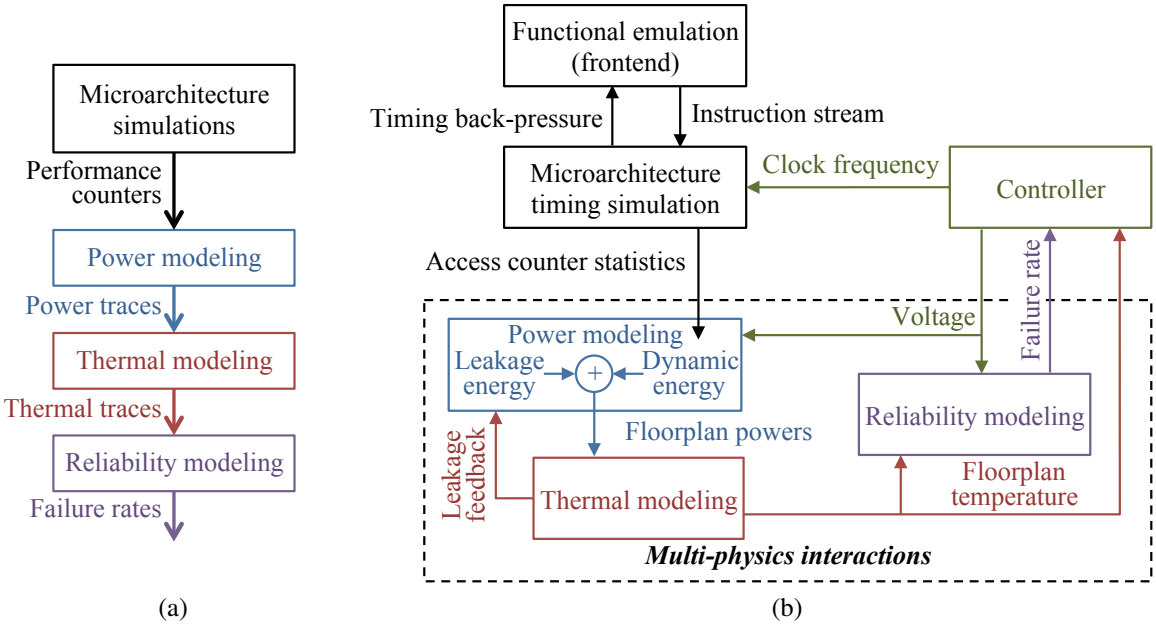


Figure 2: Comparison between (a) conventional trace-driven simulation and (b) integrated simulation with feedback interactions among applications, microarchitectures, and various physical models.

physical property is characterized at a time under the oversimplified assumptions of other physical behaviors. Such an approach does not capture interdependency and feedback interactions between multiple physical phenomena (e.g., temperature and leakage power). Ideally, we need a modeling infrastructure where interactions among all physical phenomena are captured and driven by applications executed on a target multicore microarchitecture as illustrated in Figure 2(b); the details of this integrated multi-physics simulation environment are discussed in Chapter 3.

The second challenge is the efficient integration of physical modeling tools with microarchitecture simulators, while correctly capturing the interactions among various physical properties. To date, this has normally been a laborious and error-prone engineering work. A few related efforts are found regarding the development of integrated simulation environment and multi-dimensional analysis. Coskun et al. [14, 15, 16] presented a trace-driven simulation method including power, thermal, and lifetime reliability models. Traces are the easiest way of connecting separate simulation tools, but this approach is

fundamentally limited in its ability to capture runtime dynamics. For example, adaptive controls such as dynamic frequency scaling (DFS) significantly alter time-sampled data in traces. Applying dynamic execution controls requires the artificial post-processing of pre-generated traces to reflect corresponding changes, which is difficult to process accurately and is error-prone. In contrast, a framework proposed in this dissertation (discussed in Chapter 3) takes a more holistic approach by incorporating a variety of modeling tools into the same simulation environment as the modeling flow shown in 2(b).

Bartolini et al. [5] introduced a MATLAB-based framework interfaced with a conventional C/C++-written microarchitecture simulator via copying data structures to/from shared memory space. Although MATLAB is a powerful toolkit, it is not suitable to be used in the direct integration with already time-consuming microarchitecture simulators. Sajjadi-Kia et al. [56] and Yamamoto et al. [87] developed an integrated thermal-reliability simulation framework that optimizes floorplanning. Their framework uses the detailed circuit-level information of an intellectual property (IP) block obtained from a SPICE simulation. It targets exploring different floorplanning options by varying design parameters, which is conducted in a static manner in each simulation iteration (e.g., using an average or representative power and temperature value). Priyadarshi et al. [50] presented a similar simulation infrastructure for the thermal pathfinding of 3D ICs. A major difference to the work of Sajjadi-Kia and Yamamoto is that the authors used simplified system description instead of using too detailed and therefore slow computer-aided design (CAD) models. Their infrastructure also implements a metric-driven tool flow to find an optimal 3D design by changing design parameters and evaluating performance, power, and temperature effects of each design option in a static manner.

The proposed framework in this dissertation is distinguished from these efforts in that it models workload-driven runtime dynamics in microarchitecture operations and multi-physics interactions to understand the physics and develop management techniques that can mitigate physical challenges in microarchitectural operations. In addition, flexibility

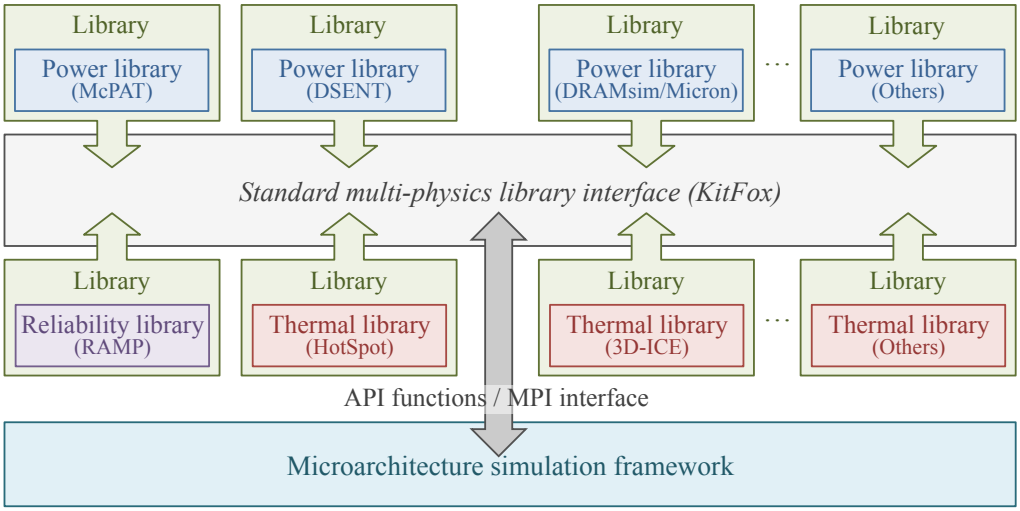


Figure 3: Library implementation that standardizes the integration of modeling tools with API to microarchitecture simulators.

in system description enables users to easily compose and simulate different processor designs, and simulations are not limited to a particular system such as 3D ICs.

Hsieh et al. [24, 25, 26] implemented a power and thermal simulation environment by integrating McPAT [40] and HotSpot [29] with Structural Simulation Toolkit (SST), a microarchitecture simulation framework [53]. McPAT and HotSpot were also integrated with the popular gem5 microarchitecture simulator [9]. In fact, these tools are connected to gem5 via traces, where the simulator generates a result file in an input format that McPAT or HotSpot can read [78]. Such a tight integration of particular point tools is not generally extensible to incorporate additional models or update existing tools to newer versions. A multi-physics framework proposed in this dissertation supports modularized integration of various modeling tools (e.g., third-party tools) via the notion of *libraries* as shown in Figure 3. This framework implements a standard API to microarchitecture simulators to ease the use of integrated models. There is no cross-dependency created between the integrated tools, and any new models can be easily added to the framework, while correctly capturing physical interactions between the tools; the details are discussed in Chapter 3.

Developing an integrated microarchitecture and physics simulation infrastructure involves numerous software engineering challenges. In particular, the simulation framework

has to be standardized, configurable, and scalable. This dissertation introduces a novel multi-physics simulation framework named *KitFox*. KitFox framework provides a standard interface and integration method to bridge various implementations of physical models. It facilitates research explorations at the intersection of microarchitecture, power, energy, thermal, cooling, and reliability. Such features and capabilities are distinguished from the state of the practice that incorporates specific point tools via tight integration with specific simulators [5, 9, 14, 15, 16, 24, 25, 26, 50, 56, 78, 87]. To the best of our knowledge, there does not exist an integrated multi-physics simulation framework that supports aforementioned features.

2.2 Architecture-Level Lifetime Reliability Modeling and Management of Multicore Processors

This section reviews two important issues of architecture-level lifetime reliability research; 1) modeling and 2) management.

2.2.1 Architecture-Level Lifetime Reliability Modeling

Modeling lifetime reliability in microarchitecture-level simulations is a challenging task. Since device aging and failure mechanisms reflect long-term behaviors, they are impractical to simulate with detailed cycle-level microarchitecture simulations for entire lifetime. It is also technically impossible to simulate billions of transistors in a processor with device-level details. Therefore, high-level abstractions are generally employed in architecture-level lifetime reliability modeling.

First, it is assumed that device-level degradation behaviors are similarly reflect in higher-level abstractions such as microarchitecture components (or processor floorplans) to reduce problem size from billions of transistors to a few tens or hundreds of microarchitecture components in the processor. Srinivasan et al. suggested this approach [71, 72, 73, 74], and it has been widely adopted in various architecture-level lifetime reliability studies [14, 15, 16, 18, 19, 24, 27, 28, 34, 37, 45, 46, 56, 85, 87]. This dissertation also uses a

similar approach for the architecture-level abstraction of lifetime reliability modeling.

Second, simplified performance, power, or thermal models are typically employed to speed up simulations to observe failures. For example, an average or representative thermal state can be used to evaluate lifetime reliability [56, 71, 72, 73, 74, 87]. This approach is mostly used for design space explorations that require assessing multiple design options instead of analyzing the details of a design. Alternatively, traces with large timing intervals (e.g., each sample representing hours or days of interval) are also often used [18, 19, 27, 28, 34, 46, 85]. Another approach is to utilize detailed microarchitecture and physics simulations instead of using simplified performance, power, or thermal models. Failure rates are calculated during relatively short time of simulation (e.g., several seconds of execution in real time, as opposed to years of time to failure) based on detailed physics modeling. The calculated failure rates are used to predict lifetime and assess the relative reliability impact of corresponding executions or applications [14, 15, 16, 24, 45]. This method can be used for analyzing transient behaviors or developing dynamic reliability management techniques.

In this dissertation, both approaches are used in different research topics. The latter approach (i.e., detailed multi-physics modeling) is used for studying performance and lifetime reliability tradeoff in multicore processors (described in Chapter 4) based on an integrated multi-physics simulation framework presented in Chapter 3. The former method (i.e., simplified models) is used for analyzing the lifetime reliability consequences of heterogeneous multicore processors compared to conventional homogeneous implementations to assess the effects of different parameters (i.e., processor composition, scheduling method, and core utilization), and this is discussed in Chapter 5.

2.2.2 Lifetime Reliability Management of Multicore Processors

Decreasing lifetime reliability is a growing concern. As Figure 4(a) shows a trend of decreasing lifetime reliability with continued device miniaturization [71, 72, 74], it is anticipated that significant reduction of failure rates will be required to sustain the current level of lifetime reliability for future processors [37, 71, 72, 74], Traditionally lifetime reliability was studied at device or circuit level. Reliability enhancement was achieved by adding large design margins or hardening circuits based on the worst-case operating conditions. However, this approach is becoming more challenging and costly since maximally achievable clock frequency decreases for the same design margin ratio with continued technology scaling as shown in Figure 4(b).

Another design approach is to add extra components on the die. Srinivasan et al. [73] studied the effect of structural duplication on enhancing processor lifetime reliability. It was shown that duplicating vulnerable structures could significantly enhance processor lifetime, instead of duplicating an entire core that was a much costly option. The authors claimed that different applications stressed a different set of components, but there were no particular components that were vulnerable to failures across all distinct types of applications. Huang et al. [27] analyzed the lifetime reliability impact of core-level redundancy. The authors showed that reserving a set of cores as spare cores could greatly improve processor lifetime, where the spare cores were used either in a rotation mode or to replace failed cores. However, structural duplication or core redundancy approaches significantly increase manufacturing costs and underutilize given processor resources.

As processor operations become more dynamic and complicated because of physical problems such as power, thermal, and reliability challenges and their coupled interactions, system-level analysis is increasingly important and requires the holistic view of problems encompassing applications, microarchitectures, and aforementioned physical challenges. Several researchers have proposed microarchitectural adaptation techniques for dynamic reliability management. Coskun et al. [15] studied the lifetime reliability impact of power

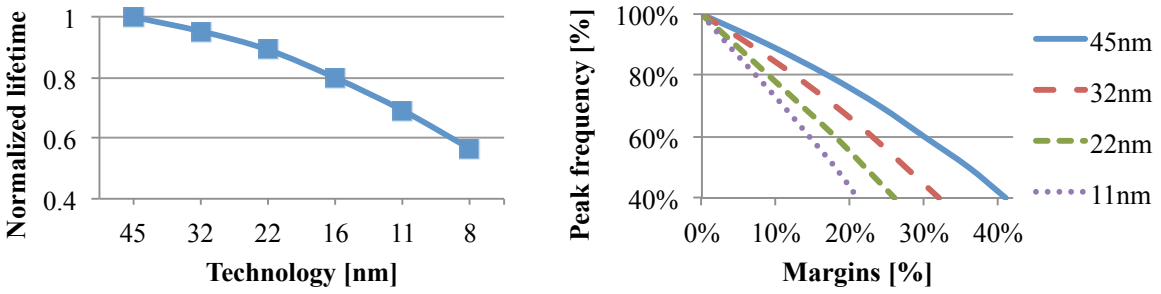


Figure 4: (a) Prediction of decreasing lifetime with technology scaling [71, 72, 74]. (b) Estimated peak frequencies with respect to design margins at each technology node [52].

management techniques and showed that power-gating of idle cores or regulating high-power operations (and thus avoiding high-temperature states) could help improve processor lifetime. Lu et al. [45] suggested a reliability banking technique that controlled processor executions to meet a specified target lifetime (e.g., 10 years). When a processor is underutilized, it is in a reliability banking (or saving) mode, where the expected lifetime of the processor is greater than the target lifetime. The saved lifetime is used by temporarily allowing high-temperature operations than a target lifetime-equivalent temperature when applications demand high performance. Karl et al. [34] proposed a similar idea to draw more performance by accelerating executions when processors were underutilized. The authors pointed out that the actual degradation behaviors of processors induced by real applications were far from the worst-case conditions or designed lifetime, and therefore there was reliability headroom to boost processor executions and trade performance with over-provisioned lifetime reliability. Mercati et al. [46] attempted to manage lifetime reliability by regulating voltage and thermal histories that had strong correlation with device aging phenomena. In these studies, lifetime reliability was treated as a finite resource, so the adaptation techniques focused on time-domain management to control how given (or designed) processor lifetime was consumed over time. These approaches well applied to traditional single-core systems, but managing spatial degradation distribution also becomes critical when the problem comes to multicore designs.

Coskun et al. [16] studied the lifetime reliability impact of different power or thermal management techniques in a multicore processor. The authors claimed that simply regulating the peak power or temperature of individual cores could be inefficient, and therefore management techniques should also consider power and thermal variations on the multicore die to mitigate their impact on processor-level lifetime reliability. Feng et al. [18] presented a workload scheduling method for wear-leveling of a multicore processor. In their scheduling scheme, wear-leveling was achieved by giving thread scheduling priorities to the weakest cores, and these cores picked the most favorable threads based on thread-level profiles and their degradation status. The previous studies commonly relied on microarchitectural heuristics to tackle lifetime reliability problems. However, they did not characterize how applications created degradation variations in multicore processors and how such variations affected system-level properties such as performance and lifetime reliability tradeoff.

This dissertation analyzes the fundamentals of application-induced degradation distributions on a multicore die and their impact on processor-level lifetime reliability. In particular, degradation distributions are characterized by using probabilistic distribution models. These models reveal that the variance of degradation distribution is a critical factor to determine processor lifetime, and therefore reducing the variance effectively leads to processor lifetime enhancement. In addition, this research addresses an important fact that performance and lifetime reliability are inversely related, and a new metric is introduced to quantify this tradeoff. It is demonstrated from integrated microarchitecture and multi-physics simulations that performance-reliability tradeoff can be improved via variance-aware reliability management using adaptation techniques such as thread migration, dynamic voltage scaling, or turbo-mode execution. The details are discussed in Chapter 4.

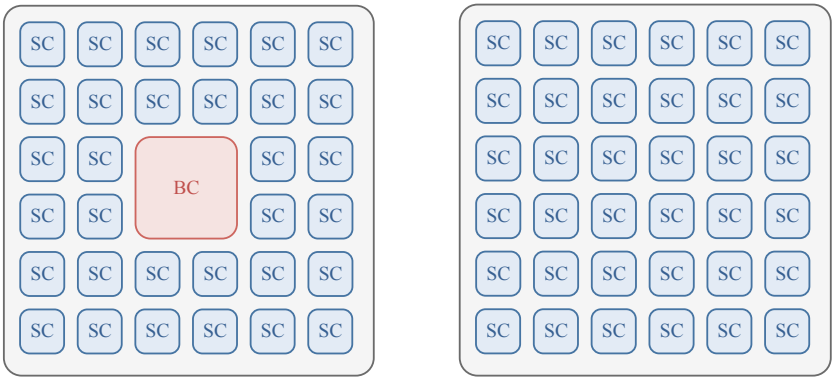


Figure 5: Multicore configurations: (a) heterogeneous processor with a big core (BC) and many small cores (SC) and (b) homogeneous processor comprised of small cores.

2.3 Amdahl's Law for Performance and Energy Scaling of Heterogeneous Multicore Processors

Heterogeneous multicore processors have been suggested as alternative microarchitectural designs to enhance performance and energy (or power) efficiency. For instance, a processor comprised of a complex core (i.e., out-of-order execution) and many simple cores (i.e., in-order execution) such as Figure 5(a) can enhance these metrics by using the complex core for faster sequential executions and many simple cores for energy-efficient parallel operations. The energy efficiency and performance improvements of the heterogeneous processor over a conventional homogeneous processor are governed by Amdahl's Law [4] as widely studied in prior work [12, 13, 17, 23, 30, 31, 38, 39, 47, 75, 83].

Applications exhibit different performance and energy behaviors depending on executing core types. Kumar et al. [39] showed that matching computing resources to application characteristics could enhance energy efficiency. Hill and Marty [23] extended Amdahl's Law to study the performance impact of a heterogeneous multicore processor shown in Figure 5(a), compared to a conventional homogeneous processor drawn in Figure 5(b). Following the work of Hill and Marty, Woo and Lee [83] presented the energy and power scaling models of multicore processors including a heterogeneous design composed of a complex core and many simple cores. Their models enabled the evaluation of energy and

power-related metrics such as performance per Joule or Watt for various multicore compositions. Chung et al. [13] explored hypothetical heterogeneous computing models comprised of conventional high-performance cores, minimally sized baseline cores, and diverse unconventional computing units such as FPGAs, GPGPUs, or custom logics. Their analytical models showed that these unconventional cores could provide better energy efficiency than conventional CPUs for highly parallel applications. As total power becomes a critical limitation, Morad et al. [47] studied the performance impact of heterogeneous multicore processors under power budget constraints. Esmaeilzadeh et al. [17] showed the projection of power-limited multicore systems with the view of emerging dark silicon era.

Using heterogeneous computing units requires sophisticated scheduling methods to maximize utilization and performance. Since different types of computing units have distinct computing capabilities, Koufaty et al. [38] suggested a method for bias scheduling to handle performance imbalance between heterogeneous cores. Suleman and Joao et al. [30, 31, 75] studied identifying critical threads in parallel executions and executing them on high-performance cores to speed up the overall execution. Cao et al. [12] presented measurement-based analysis to support virtual machine services in heterogeneous processors and improve their performance and energy efficiency. This dissertation does not dive into the details of these scheduling problems. Instead, it is assumed that these efforts would potentially enable heterogeneous multicore microarchitectures to maximally utilize computing units.

Heterogeneous processors have been largely studied in performance and energy (or power) aspects, but their reliability consequences have been overlooked. As summarized in Section 2.2, previous studies focused on characterizing and managing the lifetime reliability of homogeneous multicore systems. The proposed techniques encompass wear-leveling [18, 19, 27], component redundancy [27, 73], and thermal and power management methods [14, 15, 16, 45, 46]. A greatly simplified theoretical model is found in the work

of Huang et al. [28]. The authors studied the lifetime reliability of a heterogeneous processor comprised of a few cores, but they did not include enough details of heterogeneous multicore designs and operations. Yu et al. [88] presented a multi-dimensional analysis of homogeneous multicore processors. Their study explored various multicore compositions in terms of core size and count under the area constraint, and the performance, yield, and lifetime reliability of each homogeneous option were evaluated. This dissertation proposes theoretical performance, energy efficiency, and lifetime reliability models of heterogeneous multicore processors based on Amdahl's Law, and their consequences are discussed.

CHAPTER III

MODELING AND SIMULATION METHOD OF INTERACTING MULTICORE PROCESSOR PHYSICS

Modeling and simulation of future microarchitectures require more than performance measurement and estimation. Microarchitectural analysis must be *holistic* including energy, power, thermal, cooling, and reliability concerns since these physical constraints and their coupled interactions have become critical determinants of processor operations and performance. This requires a modeling and simulation environment that incorporates multiple physical phenomena and their concurrent interactions with microarchitecture as shown in Figure 2(b). In this dissertation, interactions between diverse physical phenomena are referred to as *multi-physics* interactions. To address this important modeling challenge, a novel, open-source modeling framework named *KitFox* [66, 69] (formerly called Energy Introspector [63, 64]) was developed and released. This framework has been successfully integrated with different multicore simulation infrastructures including Manifold [81] and MacSim [36], and it is portable across different microarchitecture simulation infrastructures. KitFox was applied to a range of research problems requiring the integrated microarchitecture and multi-physics simulations [2, 3, 41, 51, 61, 62, 65, 67, 68, 86]. In this chapter, the design methodology, implementation, and usage of KitFox framework are explained, and software engineering challenges to develop a multi-physics simulation infrastructure are discussed.

3.1 Design Motivation and Contributions

Development of KitFox framework was motivated primarily by two issues. First, there already exist a variety of point tools popularly used in the architecture community and other

custom or proprietary models in industry. Considerable efforts have been already invested to develop, validate, and release these models. Utilizing them in multi-physics simulations is a pragmatic, cost-effective, and convenient start rather than re-investing resources to develop new models of similar capabilities and features. Furthermore, the architecture community continues to develop new models or update existing tools as technology evolves. Therefore, it is desirable for a modeling framework to support the integration of various implementations of tools and be open to the easy integration of new or updated models as they are developed and become available.

The second challenge is the efficient integration of these models with microarchitecture simulation infrastructures while capturing multi-physics interactions between the models. To date, this required tedious, laborious, and error-prone engineering efforts. Therefore, this research emphasizes that the *standardization* of modeling interface and integration is a logical approach. The modeling interface should take the form of an API that is standardized across various model types and invocations. Such a framework then becomes portable across microarchitecture simulation infrastructures with only engineering efforts for porting the API instead of re-integrating all necessary physical models into each detailed microarchitecture simulator.

Architecture-level multi-physics simulations utilize various types of physical models. KitFox is based on the integration of modeling tools that simulate different physical properties; reliability, power, and thermal models including the popular tools such as McPAT [40] and HotSpot [29]. Each type of model in KitFox is encapsulated into a standard class called *library* (i.e., energy, thermal, or reliability library), and the model becomes a subclass of the library. Any new or updated models can be seamlessly integrated into KitFox by encapsulating them into the respective libraries. There are no hidden software dependencies created between libraries, and all interactions are explicitly managed within KitFox via a standard library interface. Importantly, this approach avoids modifications to external models (i.e., third-party tools).

KitFox manages multi-physics interactions. Microarchitecture simulators themselves do not need to incorporate and handle the physical models. Interface to microarchitecture simulations is implemented as a user API, which relieves the modelers from orchestrating complex multi-physics interactions within functional microarchitecture simulations. Figure 3 illustrates the concept of standard libraries and user API to microarchitecture simulators. In sum, KitFox has the following features and contributions:

- *Standard libraries*:

  Each model is encapsulated into a standard class called library and integrated into KitFox. No cross-dependency in software integration is created between the integrated models, and any new or updated models can be seamlessly added to the framework.

- *Interacting library models*:

  Interactions between the physical models are orchestrated inside KitFox via a standard library interface. It minimizes user involvement in data management to coordinate multiple libraries and subclass models.

- *Error detection and correct time synchronization*:

  KitFox provides error detection methods to ensure the correct invocation of physics calculations. At every calculation, KitFox checks if shared data (e.g., counter, voltage, power) are updated in a timely manner along with simulation progresses. Correct time synchronization and error detection avoid the misuse of integrated physical models.

- *Simple, standardized user API*:

  KitFox provides a set of user API functions to be used in microarchitecture simulations. The same function is used for calculating the same physical property via the standard library interface, regardless of which individual model is used inside the framework. For instance, all thermal models (e.g., HotSpot [29], 3D-ICE [70]) can be invoked the same way from a microarchitecture simulator.

- *Portability*:

  Since the physical models integrated in KitFox are driven by the user API functions, the entire set of models and KitFox are portable across microarchitecture simulation infrastructures with only engineering efforts for porting the API to the simulators.

- *Configurable processor description*:

  KitFox provides a configuration method that users can define the physical hierarchy of a processor (e.g., packages, floorplans, and microarchitecture components) and associate individual physical models with constituent processor components to be simulated.

- *Parallel simulations via MPI implementations*:

  KitFox supports parallel simulations via message passing interface (MPI) implementations. A microarchitecture simulator and KitFox can execute in parallel in separate MPI processes, or KitFox itself can also be divided into multiple MPI ranks.

3.2 Library Integration of Physical Models

KitFox framework provides a standardized method to integrate various implementations of physical modeling tools. An individual tool in general is specialized to model a single type of physical property, and different tools have different functionalities and usages. KitFox defines a class called *library* that hosts different type of physical models; energy, thermal, or reliability library.

For each model being integrated into KitFox, a *wrapper class* is created. The wrapper class is defined as the subclass of one of the standard library classes listed in Table 1. It includes the header and source files of the tool to be integrated, and the usage of the model is re-defined according to the virtual functions of the corresponding library class. As a result, models of the same library type can be invoked in an identical way. For instance, HotSpot [29] and 3D-ICE [70] are popular tools used for package-level thermal field calculations. Both models are integrated as thermal library in KitFox and therefore can be driven by the same functions, even though their implementations are different. When KitFox is used

Table 1: Standard Library Classes and Integrated Models in KitFox

Libraries	Description
Energy library	• Estimating per-access energies of different access types (e.g., read, write, logical switching) • Area estimation based on circuit-level models • Runtime update of variables (e.g., voltage, clock frequency) • Integrated models: McPAT (extension of Cacti) [40, 77], DSENT (extension of Orion) [32, 76], DRAMSim [54], and IntSim [57]
Thermal library	• Floorplanning and power grid mapping • Calculation of steady-state or transient temperatures • Runtime update of variables (e.g., coolant flow rate of microfluidic cooling, ambient temperature) • Integrated models: HotSpot [29], 3D-ICE [70], a compact microfluidic cooling model in 3D ICs [80]
Reliability library	• Calculation of failure rates based on time-varying stress conditions • Runtime update of variables (e.g., temperature, voltage) • Integrated models: RAMP-like wear models including electromigration (EM), bias temperature instability (BTI), time dependent dielectric breakdown (TDDB), hot carrier injection (HCI), stress migration (SM), and thermal cycling (TC) [1, 37, 71, 72, 74, 82]

in a microarchitecture simulator, the choice of one or the other model can be made with no changes to the simulator; the choice of a model is specified in an input configuration file. Table 1 summarizes the functions of library classes, and the following sections explain individual libraries.

3.2.1 Energy Library and Circuit-Level Modeling

A power or energy modeling tool is integrated as the subclass of energy library. Power is characterized at microarchitecture-level components whose circuit-level designs are estimated by supported tools based on technology-dependent parameters and microarchitectural configurations. Simulated microarchitecture is decomposed into individual components, where the models of energy library can estimate per-access dynamic energy of distinct access types (e.g., read, write, logical switching) and leakage power [32, 40, 54,

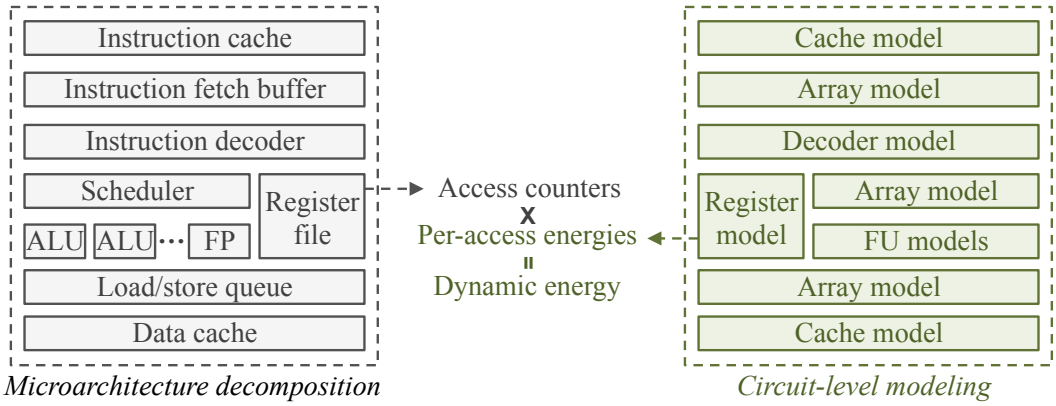


Figure 6: Architecture-level modeling of dynamic energy calculations for decomposed microarchitecture components.

76, 77] as illustrated in Figure 6.

Total dynamic energy is calculated by multiplying estimated per-access energies (E_a) with access counters (C_a) of corresponding types that can be collected from microarchitecture simulations [68], as expressed in Eq. (1). For example, to estimate the power dissipation of a register file such as the one shown in Figure 6, read and write access counters of this component are collected from a microarchitecture simulation. Per-access energy of a read or write operation is estimated by a selected circuit-level model.

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{leakage}} = \frac{\sum_{a \in \text{access types}} (C_a \times E_a)}{t} + P_{\text{leakage}} \tag{1}$$

Dynamic power (P_{dynamic}) is calculated as the sum of the products of access counters and per-access energies for each access type, divided by timing interval (t) that the counters have been collected during the microarchitectural simulation. Leakage power (P_{leakage}) has exponential dependency on temperature, so it requires a thermal analysis for accurate power modeling. While KitFox framework supports several popularly used power modeling tools in the architecture community, integration is not limited only to these tools, and new models can be seamlessly added by encapsulating them into the energy library.

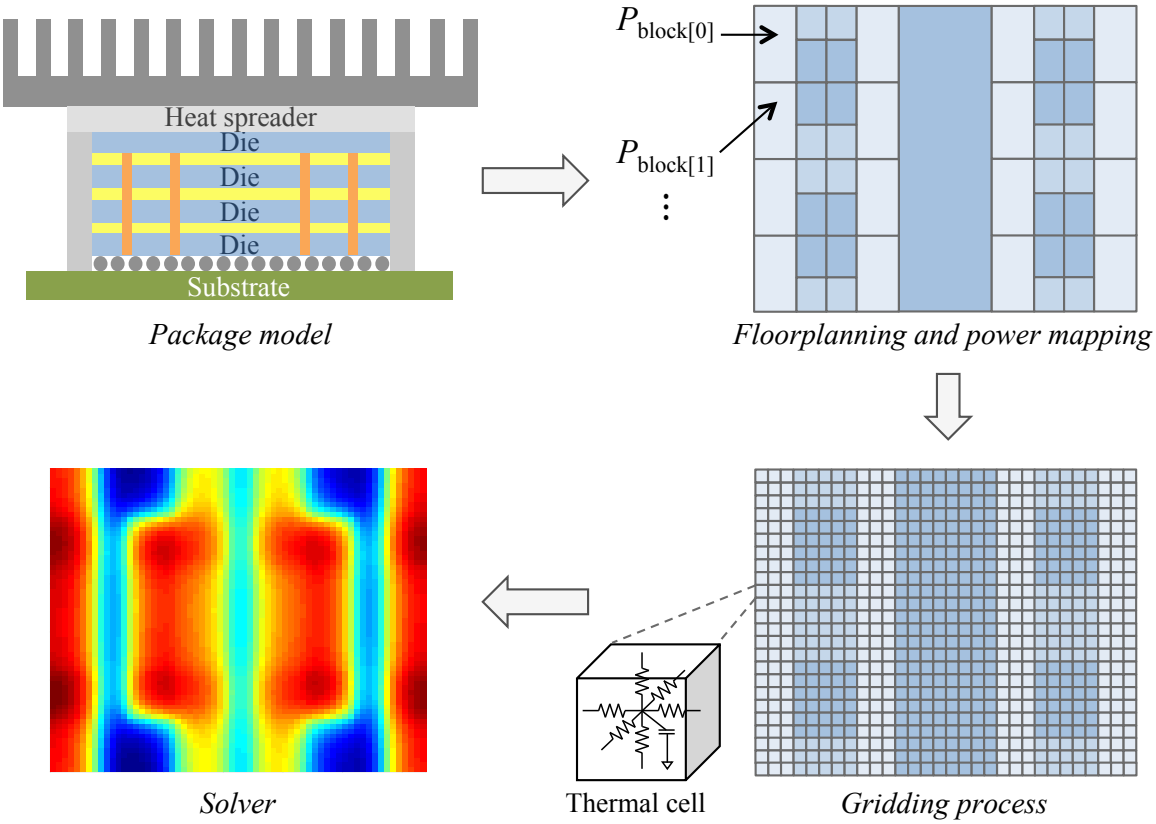


Figure 7: Package-level thermal calculation via floorplanning, power mapping, gridding, and solving thermal RC grid.

3.2.2 Thermal Library and Package-Level Modeling

Temperature modeling tools are integrated as the subclass of thermal library. Temperature is characterized at package level through a modeling flow illustrated in Figure 7. A processor package is expressed as the stack of layers with thermal grids. Each thermal grid cell is represented as a thermal resistor-capacitor (RC) connection. Calculating thermal field over a processor package is equivalent to solving the differential equations of this thermal RC network. Components constructing a processor on a die are organized and represented by floorplans [29, 70]. Each floorplan block represents a set of microarchitectural units, where the power dissipation of each unit is estimated by a linked energy library model and microarchitecture timing simulation. Power estimates of floorplans are supplied to a gridding process that calculates the power of each cell at a finer resolution (e.g., $0.1\text{mm} \times 0.1\text{mm}$

Table 2: Failure Models and Parameters of Reliability Library

Failure types	Description and model
Hot carrier injection (HCI) [37, 82]	Particles that gain sufficient kinetic energy overcome the barrier to gate oxide and cause degradation. $\lambda_{\text{HCI}} = \alpha \times {V_{ds}}^{n} \times e^{-E_a/kT}$ α = process-dependent scaling factor, $n = 3$, $E_a = -0.1$, k = Boltzmann's constant, T = absolute temperature
Electromigration (EM) [71, 72, 74]	Directional transport of electrons in interconnect wires causes degradation and failure. $\lambda_{\text{EM}} = \alpha \times J^{n} \times e^{-E_a/kT}$ J = current density, $n = 2$, $E_a = 0.9$
Negative bias temperature instability (NBTI) [37, 82]	PMOS devices under negative gate voltage at elevated temperature cause threshold voltage shift and timing error. $\lambda_{\text{NBTI}} = \alpha \times {V_{gs}}^{n} \times e^{-E_a/kT}$ $n = 5$, $E_a = 0.4$, V_{dd} = supply voltage
Stress migration (SM) [71, 72, 74]	Differences in the expansion rates of metals cause stresses. $\lambda_{\text{SM}} = \alpha \times (T_0 - T)^{n} \times e^{-E_a/kT}$ $T_0 = 500$, $n = 2.5$, $E_a = 0.9$
Time-dependent dielectric breakdown (TDDB) [71, 72, 74]	Wear of gate oxide leads to short between gate and substrate. $\lambda_{\text{TDDB}} = \alpha \times {V_{gs}}^{c(a+bT)} \times e^{(x+y/T+zT)/kT}$ $a = 78$, $b = -0.081$, $c = 0.1$, $x = -0.759$, $y = 66.8$, $z = 8.37e^{-4}$

per cell). The grid size is configurable within a thermal model. The fine-grained power grid is the input to the thermal model, and temperature field is calculated over this grid. Temperature changes are coupled to leakage power and create a feedback loop between the energy and thermal libraries.

3.2.3 Reliability Library and Lifetime Prediction

Gradual device degradation leads to permanent failures. Aging phenomena studied in this dissertation include hot carrier injection (HCI), electro-migration (EM), negative-bias temperature instability (NBTI), stress migration (SM), and time-dependent dielectric breakdown (TDDB). Examples of failure models and parameters used in KitFox are listed in Table 2. Since these failure mechanisms reflect long-term behaviors, they are impractical

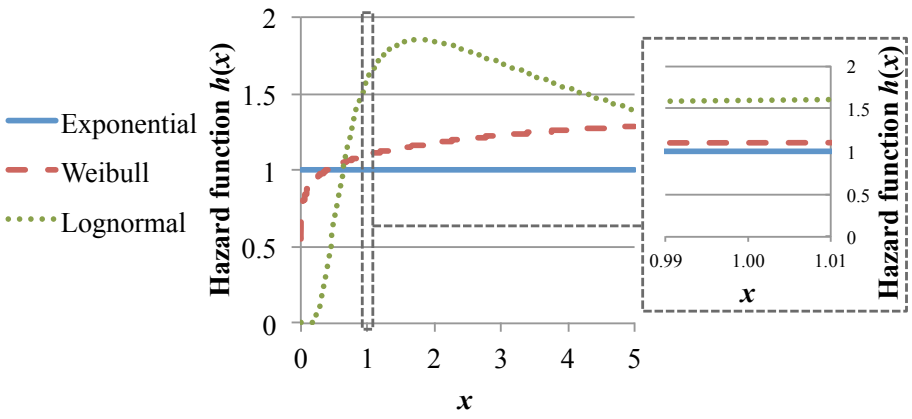


Figure 8: Effect of selecting reliability distribution functions in a cycle-level microarchitecture simulation. In the relatively short microarchitecture simulation, it becomes a constant failure-rate modeling.

to simulate across the entire processor with device-level details at cycle level. Thus, high-level abstractions are normally employed in architecture-level lifetime reliability modeling as explained in Section 2.2.1. It is assumed that device-level degradation behaviors are similarly reflected in higher-level abstractions such as microarchitecture components or floorplan blocks. Such a high-level abstraction reduces the problem size to a manageable level (e.g., from billions of transistor to a few tens or hundreds of microarchitecture components in a processor). This approach has been widely used in various architecture-level lifetime reliability studies [14, 15, 16, 18, 19, 24, 27, 28, 34, 37, 45, 46, 56, 85, 87], and KitFox uses a similar approach.

KitFox utilizes interacting physical behaviors to calculate failure rates and predict lifetime. Cumulative failure rates are calculated with respect to time-varying stress conditions including voltage and thermal stresses that are induced by workload dynamics and microarchitectural operations. KitFox currently uses common exponential models to express the failure rates (λ) of degradation mechanisms listed in Table 2. Although Weibull or log-normal distribution is known to better represent long-term degradation behaviors (e.g., in years), it can be simplified to a constant failure-rate modeling in a cycle-level microarchitecture simulation that typically spans over several seconds in real time as shown in Figure

8; selecting different distribution models may require different initial conditions. Relative changes in failure rates and their projection to mean time to failure (MTTF) can be used to assess the reliability criticality of different applications or operation modes (e.g., turbo-mode executions).

With R distinct failure mechanisms, the failure rate of a component is expressed as Eq. (2) that is the weighted average of all $\lambda_{r\in R}$. Since the relative criticality of different failure mechanisms is not known, it is assumed that these failures are equally likely at a baseline condition [14, 15, 16, 19, 27, 71, 72, 74, 85]; $p_r = R^{-1}$, where R is the number of failure models. MTTF curves with different voltage and temperature are plotted in Figure 9, where the baseline condition (i.e., normalized MTTF = 1.0) is defined at $T = 65°\text{C}$ and $V = 0.8\text{V}$ in this example.

$$\lambda = \sum_{r\in R} p_r \lambda_r \quad \text{and} \quad \sum_{r\in R} p_r = 1 \tag{2}$$

In multi-physics simulations, the failure rate λ_r of each degradation model changes over time due to temperature and voltage variations. Total failure rate at $t = t_n$ with t_i time steps ($i = 1, 2, ..., n$) is expressed as Eq. (3). $\lambda_{(t=t_n)}$ in this equation denotes the failure rate based on the λ trends up to $t = t_n$. MTTF due to such trends is calculated as $\text{MTTF} = 1/\lambda_{(t=t_n)}$, where the failure rate reflects operation history between $(t_0, t_n]$. Failure rates are affected by time-varying thermal and voltage states as shown by the models in Table 2. Hence, using an average temperature may underestimate the failure rate especially when applications or operations have large variations, since high-temperature phases have stronger impacts on aging effects (i.e., accelerated degradation).

$$\lambda_{(t=t_n)} = \sum_{i=1}^{n} \left\{ \sum_{r\in R} p_r \lambda_{r,(t=t_i - t_{i-1})} \times \frac{(t_i - t_{i-1})}{t_n} \right\} \tag{3}$$

If the exponential distribution model is replaced with other distributions such as Weibull or lognormal, the sum of failure rates (SOFR) method [71, 72, 74] shown in Eq. (2) and (3) does not hold true since these distributions have different mathematical properties. Due to mathematical complexity, using these distributions requires Monte Carlo simulations to

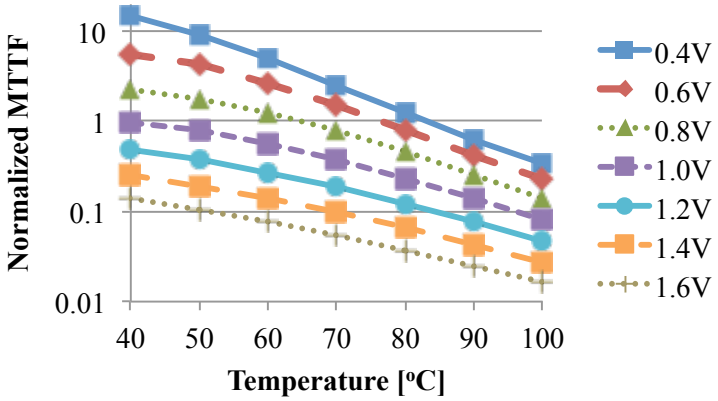


Figure 9: Lifetime reliability (MTTF) dependency on voltage and temperature based on the wear models in Table 2.

estimate failure rates and resulting MTTF [19, 73, 85]. This is a possible improvement to the reliability library of KitFox framework.

In multi-physics simulations, reliability library is coupled with thermal and energy libraries through physical interactions chain. Based on the architecture-level modeling of lifetime reliability and multi-physics interactions, KitFox enables us to explore more complex research problems such as understanding the reliability criticality of different applications or dynamic execution controls (e.g., turbo-mode executions). For example, if core execution is accelerated by elevating voltage and clock frequency, it increases switching activities of core components and voltage stress. Increased power and heat dissipations accelerate device degradation processes. As a result, the failure rate of the core rises, and predicted lifetime decreases. Such interactions cannot be correctly captured without detailed multi-physics modeling.

3.3 Processor Physics: Architecture-Level Abstraction of Application, Microarchitecture, and Multi-Physics Interactions

Interactions among diverse physical phenomena and their impact on multicore processor performance are referred to as *processor physics* in this dissertation. Modeling the processor physics is an important but challenging task. Execution controls for system-level tasks such as power or thermal management create spatiotemporal variations of physical

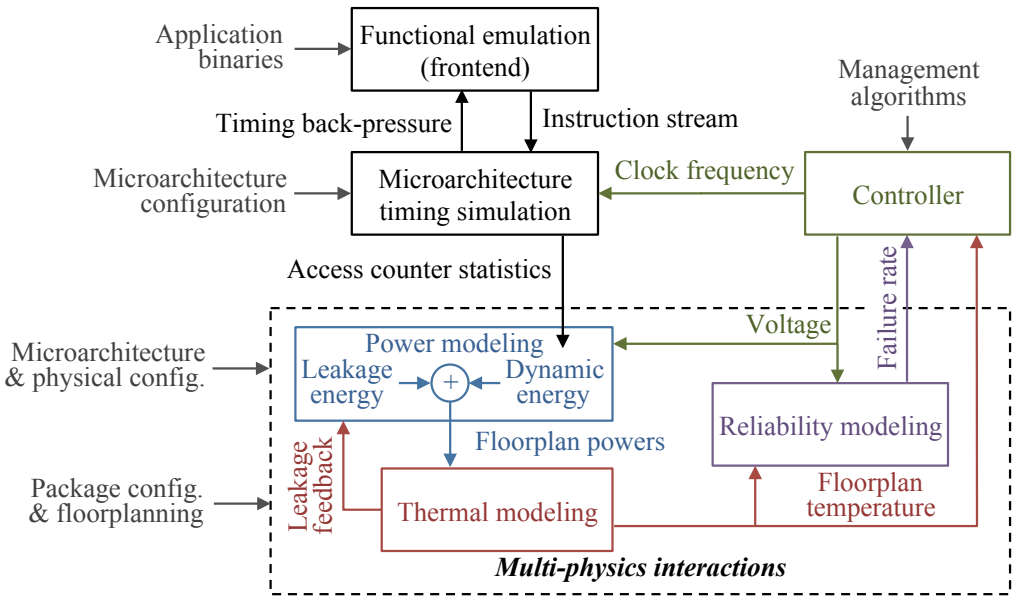


Figure 10: Architecture-level abstraction of multi-physics interactions in KitFox. Multiple distinct physical models are concurrently simulated, and interactions between them are captured during simulation runtime.

phenomena in multicore processors in addition to that created by executed applications. It is impossible to model such interacting physical phenomena with the first-order analysis such as steady-state or trace-driven simulations, where a single physical property is characterized at a time under the oversimplified assumptions of the behaviors of other physical effects. Therefore, developing an integrated multi-physics simulation environment is an imminent modeling challenge for research explorations across applications, microarchitectures, and multiple physical phenomena.

In KitFox framework, multiple physical models are concurrently simulated, and their interactions are captured at user-defined sampling rate. At every sampling interval, KitFox is invoked by a cycle-level microarchitecture simulator. Figure 10 depicts an architecture-level abstraction of interactions between multiple physical properties and associated models. Execution of workloads in the microarchitecture simulation generates switching activities of functional components (e.g., register file). Architecture activities are represented as access counters (e.g., read, write) and used to calculate the dynamic power dissipation of

modeled components [32, 40, 54, 76, 77]. Leakage power is estimated by assuming a constant temperature during a sampling interval. Since thermal response is slower than input power variations, assuming a constant leakage power is a reasonable approximation when the sampling interval is smaller than a thermal constant (e.g., in the order of milliseconds). Even if the sampling window is large, Newton's method indicates that leakage power and temperature will converge [51] in subsequent intervals rather than magnifying the gaps.

Power results are mapped onto user-created thermal floorplans, and temperature field is calculated based on the spatial distribution of input powers and thermal grid states [29, 70]. Thermal changes cause feedback interactions between temperature and leakage power, and recalculated leakage power affects temperature calculations in the next sampling period. Reliability characteristics (i.e., failure rate and lifetime prediction) of modeled components are calculated with respect to time-varying operating conditions including voltage and temperature variations. The chain of physical interactions creates a loop and is repeated at every sampling interval. Execution controls such as dynamic voltage and frequency scaling (DVFS) may intervene in this chain of events and dynamically change operating conditions and resulting physical phenomena.

3.4 Data Manipulation and Errors

In KitFox, *data queues* are used to store calculated results from libraries (i.e., physical models). Organization and operation of the data queues are central to the correct modeling of multi-physics interactions. The design, implementation, and operation of data queues are described in this section.

3.4.1 Data Formats

When integrating multiple physical models especially third-party tools in the same framework, the same physical property may be expressed in multiple distinct ways across the tools using different notations, data types (e.g., fixed vs floating point), or even units (e.g., W vs mW). There are pragmatic engineering problems to convert between different formats

of data shared across multiple tools. KitFox defines the data formats of common physical phenomena (e.g., counter, voltage, clock frequency, power, temperature), and the wrapper classes of integrated tools handle data conversions. Therefore, library classes return computed results in consistent data formats that can be shared with other libraries, and data manipulation can be standardized in KitFox.

3.4.2 Data Queues and Operations

A challenge in data management is the maintenance of time-varying data that are shared across multiple libraries and their subclass models. For instance, power is calculated by an energy library model and used to compute temperature by a thermal model. Temperature changes incur feedback interactions with leakage power, which update temperature-dependent variables of the energy model. If users or other libraries naively refer to the power data or variables stored in the energy library model, inconsistent results will be returned depending on whether the request is made before or after the calculations or feedback interactions are completed. Thus, time synchronization and retaining calculated results are essential to the correct modeling and calculation of multi-physics interactions.

In KitFox, computed results from each physical model are stored in *data queues* to handle cross-reference between libraries and runtime updates of models. Each data queue stores a single type of data (e.g., power) and is identifiable with type information that indicates which type of data is stored within. Each element in the queue is time-stamped with 1) simulation time at which it was created and 2) sampling interval at which it was recorded. Since computed results are stored in separate structures rather than overwriting the variables of models, users or other libraries can refer to the correct results with time tags while runtime updates of the models can independently proceed without corrupting the results. There are two types of queues to manage time-varying data as follows.

1. *Closed queue* for periodic data: Discrete-time data such as power or temperature are calculated and stored at the end of sampling interval, time $= t$, based on observed

statistics during period p. Closed queues store periodically sampled discrete-time data. These data are regarded as valid during $(t-p,t]$ interval, and the queue returns the data for access requests between $t-p$ and t (sec) including the time point at t (sec).

2. *Open queue* for aperiodic data: Some types of data are aperiodically collected during runtime simulation. For example, clock frequency remains constant until a control mechanism (e.g., DFS) decides to change the clock frequency. In such a case, datum stored at time $= t$ is valid for $[t,\infty)$ or until a new value is inserted at $t+\delta$ (sec). Open queues store aperiodically varying data values. The queue returns the data for access requests between t and $t+\delta$ (or ∞) including t (sec).

There are three types of operations defined for data queue accesses; *push* (data insertion), *pull* (data retrieval), and *overwrite* (data replacement). Push and pull are basic write and read operations of the queue, respectively. A pull operation does not dequeue an entry, but dequeueing is implicitly handled when the queue becomes full. The size of data queues can be defined by users in input configuration files. An overwrite function replaces an existing entry with a new value.

All queue operations require a data identifier (i.e., enumerated type) and tag information (i.e., time t and sampling period p). A queue operation first finds a data queue structure with the data identifier, where each data queue stores a single type of data. Time tag is checked at every queue operation, and error detection is provided for functional correctness and debugging purposes. For a push operation, data intervals must be contiguous, and timing violations set error codes (e.g., non-contiguous, overlapped, out-of-order) that allow users to decide whether to debug or ignore errors. If the period p is not provided (i.e., $p=0$), the queue operation implicitly derives the period value by checking the last entry in the queue; the time tag t of new data must be greater than the recorded t value of the last entry in the queue. For a pull operation, a data request must match with the time tag (t and p pair) of an existing entry. If no matching entry is found, an error code is set (e.g., tag-mismatch,

out-of-queue-range). If the period p is not provided, it tries to find an interval that the time t falls into and returns the data value of that interval. Overwrite is a combination of pull and push operations. It first performs the same process as the pull operation and then replaces the data value if a matching entry is found. Overwrite operations are useful to repeatedly update a data queue entry such as adding power numbers from multiple components.

3.4.3 Data Queues and Library Callbacks

Data queues are coupled with libraries such that inserting new data into the queues (e.g., push operation) triggers the *callback functions* of associated libraries. The callback functions perform updating the variables of library models that are dependent on the inserted data type. For instance, if a microarchitecture simulator needs to perform dynamic voltage scaling, this is simply inserting a new voltage value into the data queue associated with an energy library model through a KitFox API call. Then, the push operation triggers the callback function of linked energy library, which updates the variables of subclass power model that depend on the voltage. Similarly, library callback functions are used to handle physical interactions such as leakage power and temperature dependency. As such, runtime updates of library models can be greatly simplified and automated by managing data in the queues since transferring data between the queues implicitly incur the updates of associated library models. Specific implementations of callback functions have to be defined by the wrapper classes of individual modeling tools.

3.4.4 Error Propagation in Physical Interactions Chain

By integrating multiple physical models into the same framework, the inaccuracy of a model may propagate through multi-physics interactions chain and sometimes be amplified by other models. Although KitFox does provide the detection of timing and synchronization errors, it is important to understand how the errors of models propagate through the multi-physics interactions, thereby affecting simulation results. Although validating individual models is also an important task, the external tools [29, 32, 40, 70, 76, 77, 80]

Table 3: Error Propagation Through Physical Interactions Chain in KitFox

Power inputs		Error propagation in results	
Power density (avg. temp.)	Max input error	Resulting thermal error	Resulting MTTF error
50 W/cm^2 (68°C)	10%	1.6%	2.6%
	30%	4.9%	7.9%
	50%	7.6%	12.7%
100 W/cm^2 (90°C)	10%	2.4%	3.7%
	30%	6.9%	11.3%
	50%	11.4%	20.2%
125 W/cm^2 (101°C)	10%	2.8%	4.2%
	30%	7.7%	12.4%
	50%	12.6%	21.7%

claim to be validated to certain degrees. More importantly, they are not the contributions of KitFox, but the goal of KitFox framework is to implement a multi-physics simulation environment based on the integration of various physical models.

To study this issue, 8×8 checkerboard floorplans of 256mm^2 area are created with even power density on the die. Steady-state temperature and resulting MTTF of each floorplan are calculated under these conditions. Then, uniformly distributed errors are added to the power inputs, and corresponding changes in resulting temperature and MTTF are measured. HotSpot steady-state thermal model with default parameters [29] is used in this experiment. Failure models presented in Table 2 and Section 3.2.3 are used to estimate the MTTF, and these models are adjusted to meet 5 years of processor-level MTTF at $T = 65$°C and $V = 0.8$V operating conditions. In this experiment, it is assumed that feedback interactions between leakage power and temperature are self-contained within power inputs such that the input powers already reflect the values that temperature and leakage power converge. Therefore, thermal runaway conditions are not considered here, which require more detailed power information such as 1) dynamic and leakage power ratio in power inputs and 2) leakage power sensitivity with respect to temperature change that is highly dependent on technology parameters.

Table 3 shows the changes in resulting temperature and MTTF caused by injected errors in the power inputs. With increasing power density (e.g., 50W/cm^2 vs 100W/cm^2), the same input error magnitude produces larger absolute changes in power density and hence resulting temperature and MTTF. Temperature values in the parentheses are average steady-state temperatures at given power densities without injected errors. Due to thermal spreading effects, temperature changes are much smaller than the error rates induced in the power inputs. However, non-linearity in MTTF equations is biased against thermal hotspots, so the error rate increases with greater unevenness in power and thermal densities. In overall, this experiment reveals that errors in power inputs are at least not significantly magnifying through the physical interactions chain implemented by KitFox, and therefore multi-physics simulations can tolerate the limited inaccuracy of models.

3.5 Processor Component Hierarchy and Description

KitFox framework pursues flexibility in processor description to enable the simulation of various different microarchitectures or package designs. A principal challenge is in interconnecting multiple libraries and configuring their subclass models corresponding to a target processor design to simulate, while being able to flexibly change model parameters or modify processor designs. In KitFox framework, this is achieved by implementing a unified configuration method to define a processor component hierarchy (e.g., packages, floorplans, microarchitecture components), where each component in the hierarchy is associated with a library model.

3.5.1 Representation of Processor Component Hierarchy

In KitFox framework, a processor is represented as the hierarchy of *pseudo components*. Pseudo components are abstract units for which library models are attached to estimate physical phenomena. Different physical properties are characterized at different levels of

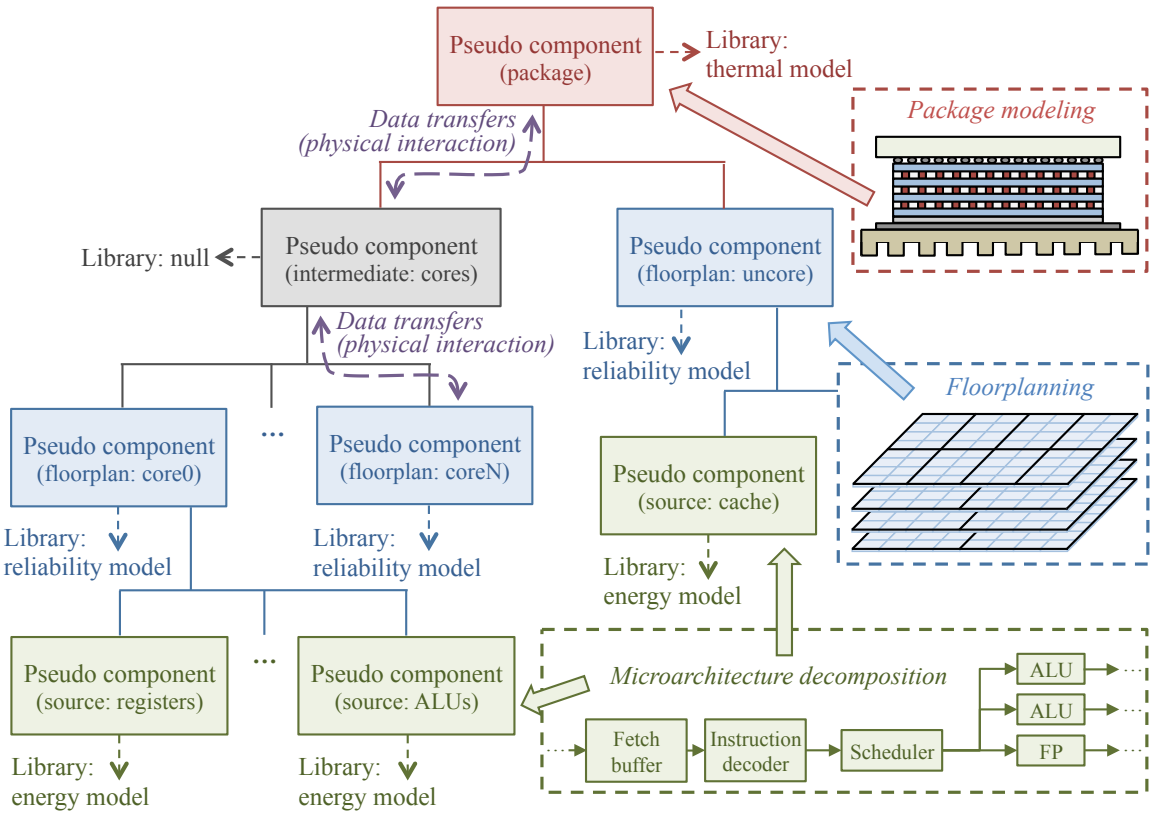


Figure 11: A processor is represented with the *pseudo component hierarchy* in KitFox. Pseudo components are physically defined units where associated libraries and their subclass models simulate physical phenomena. Physical interactions are emulated by transferring data between the data queues of pseudo components.

processor abstraction. For instance, power is characterized at microarchitecture or circuit-level components using activity counts (explained in Section 3.2.1). Temperature is calculated at package level based on power distributions on the die (described in Section 3.2.2). As such, pseudo components can represent different levels of processor abstraction depending on which library models they are associated with and which physical properties are characterized. A pseudo component may represent a microarchitecture component when an energy library model is associated with it to calculate power or energy dissipation. Or it can be a processor package if a thermal library model is attached. The pseudo component hierarchy can be flexibly composed to simulate different processor designs. There are no inherent restrictions on the number of levels in the pseudo component tree, and each pseudo component can have as many sub-components as necessary.

Figure 11 illustrates an example of how KitFox framework serves to interface pseudo components and libraries to simulate a processor design. The simulated microarchitecture is decomposed into basic components (shown as "sources" in the figure), where power is estimated by energy library models. Each energy library may derive a different tool, so it enables choosing the most appropriate model for different microarchitecture components. Pseudo components can be grouped into another upper-level pseudo component (shown as "floorplans" in the figure) depending on their microarchitectural and technological similarities (e.g., core, cache). Higher-level components may represent larger processor units such as cores or regions on the die. The root component in the example represents a processor package and is linked to a thermal library model. It can designate any descendant components in the tree as its constituent floorplans. Some intermediate components without linked libraries can also be created for the convenience of processor description or data collection. Every pseudo component includes data queues to store the computed results of library models and shared data (e.g., voltage, clock frequency, power) for cross-referencing between pseudo components.

When composing a pseudo component hierarchy, users have to know what the models of choice are capable of simulating and how they are configured. The users should specify the input parameters of each model to be used at the corresponding pseudo component. KitFox itself does not perform microarchitecture-aware automation (e.g., optimizing microarchitecture designs for system metrics such as performance or power efficiency). Technically, KitFox only recognizes the pseudo component hierarchy and libraries associated with the components. For instance, KitFox does not know if a pseudo component is representing a register file or cache but treats each pseudo component in the tree as a unit linked to one of the libraries. Microarchitecture simulators are responsible for providing complete activity statistics (e.g., access counters, timing information) with KitFox and its library models. This approach tackles a problem that developers write simulators in many

different ways. There is no common way to organize the simulation models of microarchitectural blocks into specific C/C++ functions or classes. The notion of pseudo components enables simulator users to map code segments from specific simulators to KitFox libraries, thereby making it easier to incorporate multi-physics models into any existing simulators and to do so in a portable manner.

3.5.2 Steps in KitFox Framework Executions and API Functions

KitFox framework provides a set of API functions for data calculation or manipulation that can be used in user microarchitecture simulators. A pseudo code example is shown in Algorithm 1. Microarchitecture simulation is organized as a sequence of sampling intervals (e.g., 1 million clock cycles or 1 millisecond). At the end of every sampling interval, collected access counters are used to calculate the power dissipation of modeled components, and the results are stored in the data queues of corresponding pseudo components. Power data are synchronized in the pseudo component hierarchy by aggregating the values from the leaves toward the root of the tree. The data queues of pseudo components without energy libraries (e.g., floorplans or package) are also updated based on the power values of constituent components. Since data in the queues are tagged with time information, timing violation can be detected when pseudo components have asynchronous power data or if the powers of some components are mistakenly not calculated. This synchronization process is handled inside KitFox, and any pseudo components can be probed to retrieve the power data after synchronization.

Updated power information of floorplan-level components is used to calculate temperature. A thermal library model internally converts floorplan powers to grid-level power distribution, updates thermal states, and translates grid-level thermal states to floorplan temperatures. When synchronizing temperature data in the pseudo component tree, it is assumed that temperature is uniform within each floorplan component if no further placement

```
while (program runs) do
    /* Access counters of all decomposed components are collected during
    microarchitecture simulations. */
    do (microarchitecture simulation and counters collection)

    /* At the end of sampling interval, API functions are called for physical modeling. */
    if (at the end of sampling interval) then

        /* Calculate the power dissipation of all modeled microarchitecture components.
        */
        for (all microarchitecture components) do
            kitfox->calculate_power(uarch_component_id, current_time,
            sampling_interval, counters);
        end

        /* Temperature is calculated after power calculations are done. Data
        synchronizations are internally performed. */
        kitfox-> calculate_temperature(pkg_component_id, current_time,
        sampling_interval);

        /* Failure rates are calculated with the components associated reliability library
        (see Figure 11 illustration). */
        for (all floorplan components) do
            kitfox-> calculate_failure_rate(flp_component_id, current_time,
            sampling_interval);
        end

        /* Any pseudo components can be probed to retrieve data from their queues. */
        power_t core_power;
        int err = kitfox-> pull_data(core_component_id, current_time,
        sampling_interval, KITFOX_DATA_POWER, &core_power);

        /* Voltage scaling can be done by inserting a new value into the queue and
        synchronizing pseudo components. */
        Volt core_voltage = 1.0; // 1.0V
        int err = kitfox-> push_and_synchronize_data(core_component_id,
        current_time, sampling_interval, KITFOX_DATA_VOLTAGE,
        &core_voltage);

        /* Reset access counters at the end of interval. */
        do (reset all microarchitectural access counters)
    end
end
```

Algorithm 1: A pseudo code example of KitFox API functions in a microarchitecture simulation loop.

information is provided with its sub-components. If there are multiple sub-components belonging to the same floorplan, they are updated with the same temperature. This process is technically inserting new temperature values with time tags into data queues (i.e., push operation). As a result, the callback functions of library models (e.g., energy library) are invoked, and dependent variables and states are updated (e.g., thermal and leakage power dependency). Since the calculated results are stored in data queues, library models can safely update their internal variables and states based on defined interactions with time-varying physical properties.

Lifetime reliability is characterized at floorplan-level components in the example. Cumulative failure rates are calculated with respect to time-varying stress conditions including voltage and temperature, and resulting MTTF is estimated. Instead of using one representative value (e.g., average temperature) for the entire processor, KitFox framework utilizes multi-physics modeling for reliability characterization during microarchitecture simulations. Although this approach may not give precise prediction of unknown future operations, the likelihood estimation of MTTF can be used to address the relative reliability criticality of different microarchitecture operations or applications.

After data synchronization, any pseudo components can be probed to retrieve data from queues. Dynamic execution controls such as voltage or frequency scaling can be simply applied by inserting new values into the queues and synchronizing the data in the pseudo component hierarchy. Voltage and frequency synchronizations are performed in a similar manner as temperature synchronization. All descendent components are updated with the same voltage and frequency values, and the callback functions of library models are invoked to update dependent variables. For instance, changing the voltage of a core-level component (shown as one of the floorplans in Figure 11) updates all its sub-components within the core to the same voltage.

In sum, pseudo components enable flexible composition of processor designs and interconnection of various library models. Physical interactions can be easily modeled via

data queue operations and callback functions of libraries. Although KitFox supports automated synchronization and timing error checking for easier data manipulation and correct data calculations, it does not implement optimizing system configurations with respect to particular metrics (e.g., energy efficiency). Such optimizations are realized external to the modeling environment (i.e., controller in Figure 10) by utilizing KitFox to extract physical data (e.g., power, temperature) and tune model parameters via API functions.

3.6 Parallel Interface for Scalable Simulations

Serial simulations can be time consuming when employing computationally intensive physical models over large number of components (e.g., high core-count processors). KitFox framework, developed with SST [53] and Manifold [81] parallel simulators, supports parallel simulation environment via MPI implementations. In particular, KitFox framework can run in parallel with microarchitecture simulators, or KitFox itself can run in multiple MPI processes by dividing pseudo component hierarchy into multiple parts.

There are two possible ways of supporting multi-process simulations; *push* and *pull models*. A push model requires each component of a microarchitecture simulator to call KitFox API functions in a correct sequence such as the one shown in Algorithm 1. In this case, KitFox passively responds to data access or calculation requests from the microarchitecture simulator and does not manage the simulation progress of microarchitecture models. Out-of-order invocations of API functions result in timing errors in KitFox. In a pull model, KitFox is integrated as a component of the microarchitecture simulator, and therefore KitFox processes are naturally embedded in the progress of microarchitecture simulation. At user-defined sampling interval, KitFox distributes messages to functional components through the message channel of simulation kernel and collects architectural statistics (i.e., access counters). The functional components in the simulator are only responsible for providing a complete set of counters in response to the requests, and KitFox itself handles physics calculations.

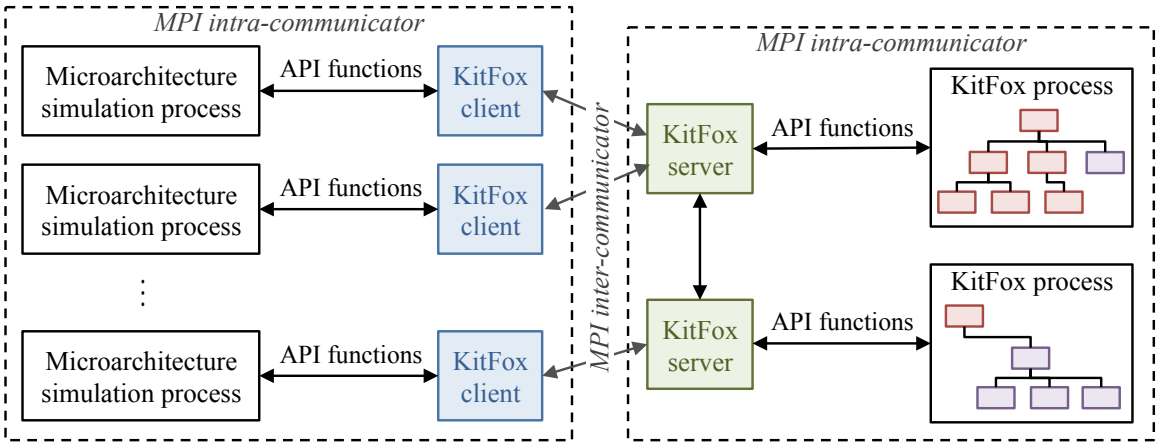


Figure 12: A *push model* example of multi-process KitFox in parallel with microarchitecture simulation processes via MPI implementations and split communicators.

Figure 12 illustrates an example of the push model. Each KitFox process initiates a *server* to handle MPI messages from/to other KitFox instances in different processes as well as client microarchitecture simulators. KitFox servers wait for MPI messages to arrive, identify message types, call necessary KitFox API functions, and return the results if necessary. Calculation functions are all non-blocking, and the microarchitecture simulation can proceed without waiting for the KitFox processes to finish calculations. However, data manipulation has to be blocking since it requires a return value (e.g., access to data in queues) for the request.

To run KitFox in parallel with an MPI-based microarchitecture simulator, the MPI communicator (i.e., a message channel) has to be split to isolate the communication of parallel microarchitecture simulation from the parallel KitFox processes. Otherwise, the microarchitecture simulator may happen to wait for the MPI barriers of KitFox processes, or vice versa. KitFox servers are fully connected with each other within an MPI *intra-communicator*, and messages from/to client simulators are through an *inter-communicator*. By dividing the pseudo component hierarchy into multiple processes, each KitFox instance only creates and initializes pseudo components that are modeled in its process. The microarchitecture simulator calls KitFox API functions through *client objects* that handle

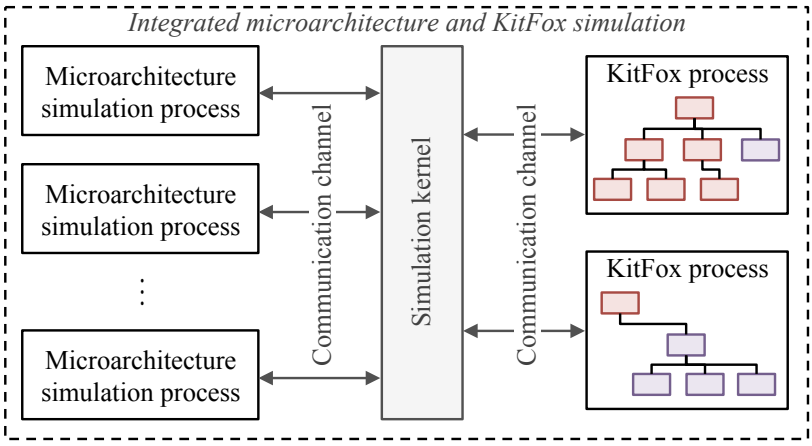


Figure 13: A *pull-model* example of multi-process KitFox embedded as a component of a microarchitecture simulator. Communications are made through the message channel of the microarchitecture simulation kernel.

message interfaces to KitFox servers. The KitFox servers remain active until all the simulator processes terminate by sending disconnect messages.

The major difficulty of using parallel simulations is in correct synchronization across parallel processes. In the push model, KitFox runs in a user-driven manner, which requires user microarchitecture simulators to invoke KitFox API functions in the correct time sequence. When the invocations occur out of order in parallel simulations, KitFox detects timing errors and prevents time-incorrect calculations. When timing errors occur in physics calculations (e.g., power calculations), KitFox prints out which pseudo components encountered timing errors and then terminates the simulations. However, if the errors are due to queue operations (e.g., access to data in queues), users may decide whether to ignore returned error codes from KitFox or terminate the simulations. Synchronization across parallel processes becomes more difficult when microarchitecture components dynamically change operating clock frequencies. The pull model may ease the synchronization problem since KitFox is integrated as a component of the microarchitecture simulator as illustrated in Figure 13. Instead of splitting an MPI communicator to isolate KitFox communications, KitFox utilizes the message channel of simulation kernel. KitFox components are processed along with microarchitecture simulation progresses, so it becomes possible for

KitFox to determine when to send requests to functional components and collect architecture counters for multi-physics modeling.

3.7 Simulation Overhead of Multi-Physics Modeling

When multiple models are simultaneously simulated, simulation speed is generally of a concern. Figure 14 shows the simulation time breakdown of an exemplary multi-physics simulation with KitFox integrated with Manifold microarchitecture timing simulator [81]. The cycle-level timing simulation of Manifold is known to be as fast as 200 kilo-instructions per second (KIPS) with a single-thread simulation, which is several times faster than highly detailed microarchitecture simulators that are generally known to run around or less than 50 KIPS of simulation speed [9, 43, 44]. McPAT [40] and HotSpot [29] are used in the measurement, which are the most popular open-source power and thermal modeling tools in the architecture community. In the exemplary simulation, 64 out-of-order cores are configured by adapting the Intel Xeon processor model of McPAT, and the transient thermal model of HotSpot is used with a 64×64 grid configuration. The pseudo component hierarchy of KitFox is built similar to Figure 11. Simulation time depends on 1) which models are selected, 2) how they are configured (e.g., number of cores), and 3) actual hardware that runs the simulation. Hence, results in Figure 14 are to show the performance of an exemplary multi-physics simulation and do not represent the optimized or the best performance that individual simulators or models can achieve.

In Figure 14(a), the interval over which counters of the microarchitecture timing simulator are sampled for calculations is varied between 100ns and 10ms. The clock frequency of simulator components is set to 1.0GHz, so 100ns corresponds to 100 clock cycles in this example. The result shows that the multi-physics simulation is primarily dominated by KitFox operations when the sampling interval is short (less than 10μs), but the microarchitecture simulation becomes the bottleneck when the interval is sufficiently long (greater than 100μs). Here, the power, thermal, or reliability portions in the figure denote

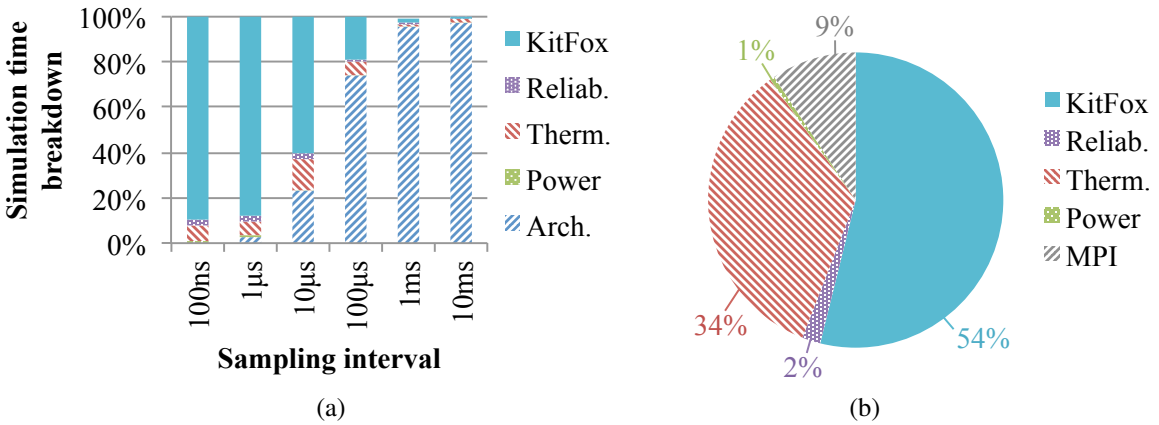


Figure 14: (a) Time breakdown of an exemplary multi-physics and microarchitecture simulation with varying sampling intervals. (b) Time breakdown of multi-physics calculations in an 8-process push-type parallel simulation at 1ms sampling rate.

only computation time excluding the handling of input parameters, status updates, or data synchronizations that are all counted as KitFox overhead. When these models are individually used, they also spend significant duration of time on handling input and output data rather than computations, and these parts are counted as KitFox operations in the integrated multi-physics simulation. In the typical range of sampling intervals (e.g., 100μs or greater), microarchitecture simulation is the bottleneck rather than multi-physics calculations. Therefore, multi-physics simulation is not as time-intensive to be the bottleneck, but developing such a multi-physics simulation environment via the integration of various physical models is a highly complicated and challenging task as addressed by KitFox.

Figure 14(b) plots the time breakdown of multi-physics calculations when the sampling interval is set to 1ms in an 8-process push-type parallel simulation via MPI. The transient thermal calculation time of HotSpot is increasing for larger sampling intervals, so the breakdown in Figure 14(b) appears different from those in 14(a) with shorter sampling intervals. The result shows that KitFox operations (mostly data manipulation and synchronization) take the most time in multi-physics calculations, and parallelization via MPI also adds non-negligible overhead to the overall simulation time. However, as shown in Figure 14(a), the simulation bottleneck is in microarchitecture simulations rather than

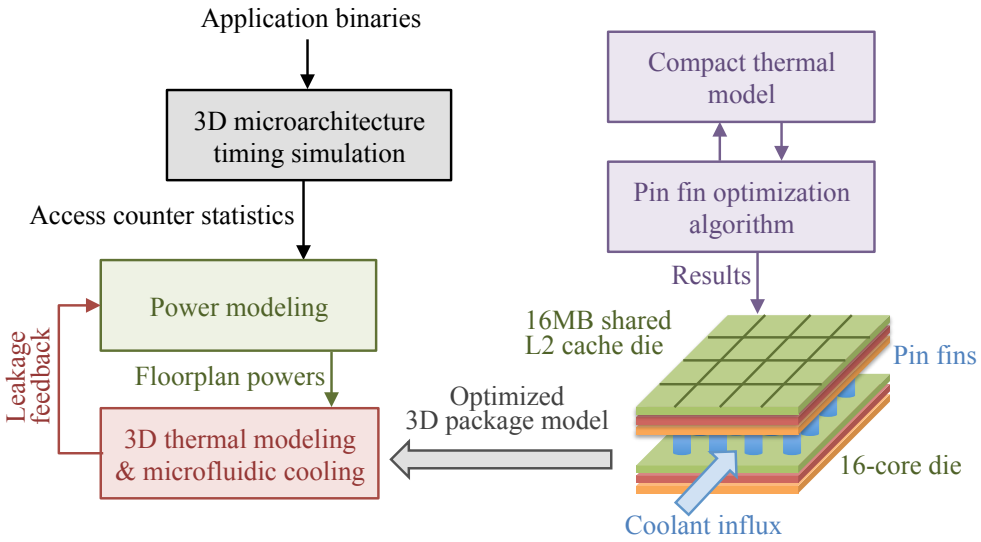


Figure 15: Simulation flow for leakage power reduction in 3D ICs via microfluidic cooling and pin fin geometry optimization [86].

multi-physics calculations for sufficiently long sampling intervals. Therefore, the main reason for parallelization is to speed up microarchitecture simulations, and KitFox should be able to support multi-process simulations.

3.8 Applications of Integrated Power, Thermal, and Reliability Simulations

This section presents the applications of KitFox framework to a range of research problems (other than those presented in this dissertation) and explains how KitFox was used to drive those studies. The breadth applications demonstrate the *versatility* of KitFox framework.

3.8.1 Leakage Power Reduction in 3D ICs with Microfluidic Cooling

3D-stacked integrated circuits improve performance and energy efficiency by shortening interconnection lengths between processing and memory entities. However, increased power density per unit volume may threaten the thermal stability of the processor. Microfluidic cooling in 3D ICs can provide superior cooling performance over conventional air-cooled packages.

Xiao et al. [86] explored the performance and energy benefits of microfluidic cooling in

a 3D-stacked processor using KitFox framework as illustrated in Figure 15. They presented an algorithm that minimized junction temperatures under given power budget by optimizing pin fin dimensions including pin diameter, height, spacing and coolant flow rate. Then, 3D-ICE thermal model [70] integrated in KitFox was configured to simulate the optimized 3D package design. McPAT [40] was used for power modeling, and interactions between leakage power and temperature were captured within KitFox framework. The authors evaluated the performance and energy impacts of the optimized 3D package with microfluidic cooling by simulating PARSEC benchmarks [8] in Manifold microarchitecture simulator [81]. In this study, KitFox facilitated the use of physical models (i.e., McPAT and 3D-ICE) and automated the modeling of multi-physics interactions, which helped the authors evaluate the optimized 3D package design in a full-system microarchitecture and multi-physics simulation environment.

3.8.2 Power, Thermal, and Throughput Regulations via Adaptive Gain Controllers

Processors are designed to sustain the worst-case operations such as thermal design power (TDP), but applications rarely operate at these limits. From a performance perspective, it is preferred for the system to be designed for average case behaviors (and therefore higher average performance) and adapt to rarely occurring extreme conditions. Rigorous control models can potentially enable such adaptive operations.

Almoosa et al. [2, 3] and Rao et al. [51] presented adaptive gain controller algorithms that regulated throughput (i.e., instructions per second), power, or temperature of multi-core processors. The proposed controller algorithms utilized core-level DVFS capability to adjust operating voltages and clock frequencies of cores to regulate throughput, power, or temperature to a reference level (i.e., desired output). A closed-loop system as drawn in Figure 16 shows how the system output can track the input reference based on Newton's method. The authors proposed control system models to regulate processor throughput, power, or temperature and demonstrated their control algorithms using KitFox. In the

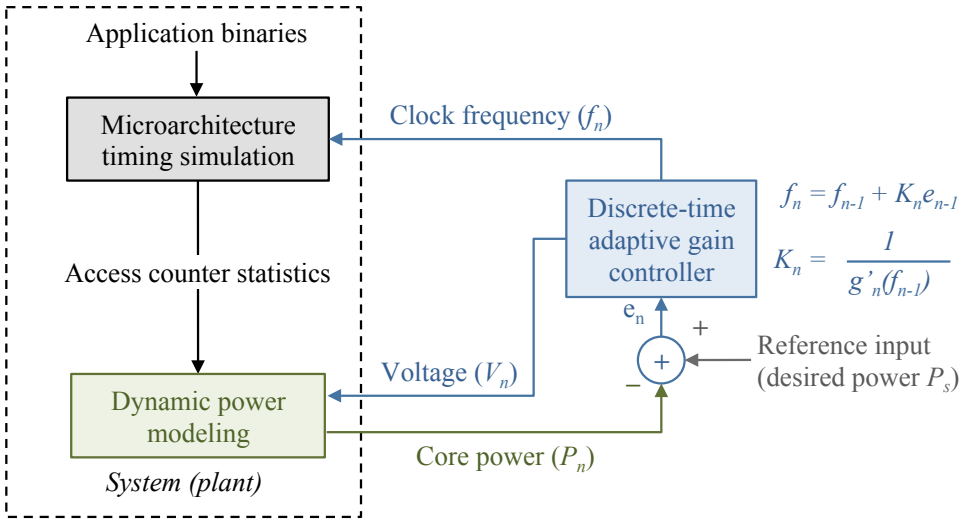


Figure 16: A control system model for processor throughput, power, or thermal regulation via an adaptive gain controller [2, 3, 51].

KitFox framework, the DVFS-based control algorithms could be easily applied via API functions, whereas these features were not supported in the original modeling tools.

3.8.3 Power Modeling of GPU Architectures

The large number of stream multiprocessors (SM) with multiple levels of memory hierarchy in GPUs collectively consume significant amount of power. Power characterization is one of the key challenges in GPU research. Considerable efforts have been invested in the past decades to develop CPU power modeling methods and models, but relatively fewer attempts are found regarding more recent needs for GPU power modeling. Although GPU microarchitecture are substantially different from CPUs, developing a new set of power models involves substantial efforts.

Lim et al. [41] approached the GPU power modeling problem by using McPAT [40], a CPU power modeling tool. The authors noted that McPAT was basically comprised of circuit-level models including caches, interconnects, latches, etc., where many of these models could be reused for GPU power modeling. However, McPAT supported only CPU-based microarchitecture designs, so the authors used KitFox to re-factor the basic circuit-level models in McPAT to configure an NVIDIA Fermi-like microarchitecture as depicted

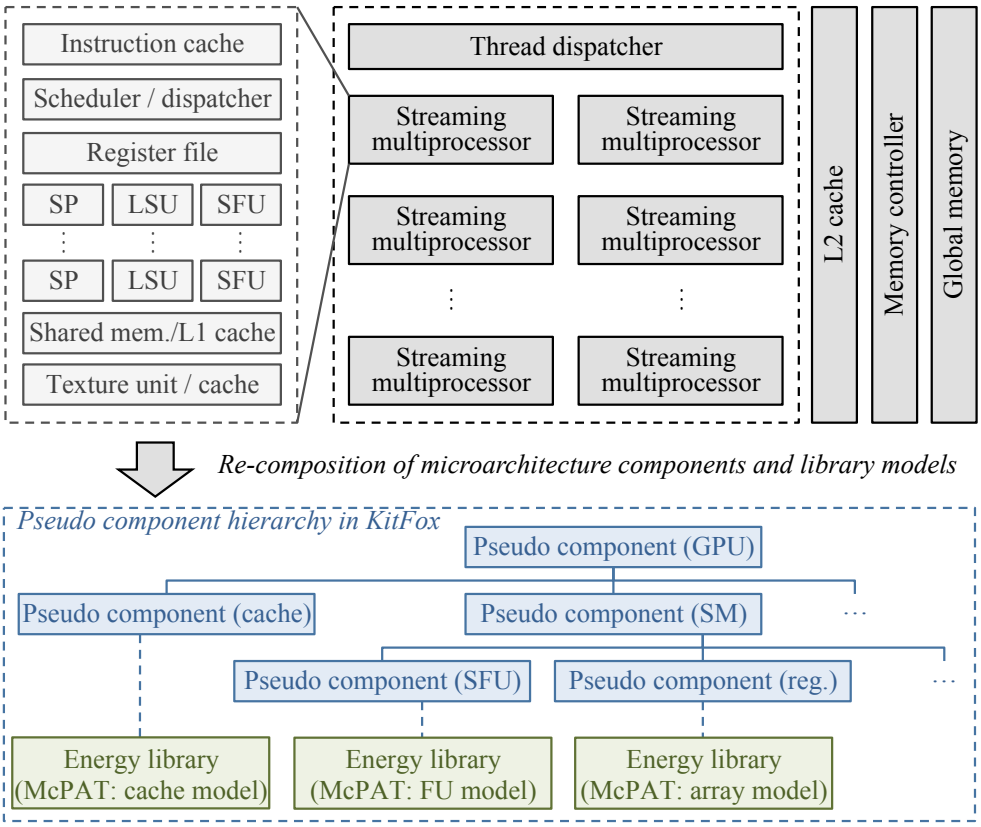


Figure 17: GPU power modeling using KitFox that provides a method to flexibly recompose microarchitecture components and related power models [41].

in Figure 17. Since KitFox interface was independent of specific microarchitecture designs, it could easily adapt to simulate a different microarchitecture while utilizing an existing power modeling tool.

3.9 Summary

Modeling interacting physical phenomena in multicore processors is becoming increasingly important. The proposed KitFox framework facilitates the use of various implementations of physical modeling tools that are integrated as libraries. The library integration method standardizes interfaces between the models and also enables future extension to other different library types or incorporation of new models. This framework enables multi-dimensional research explorations at the intersection of energy, power, temperature,

and reliability in conjunction with microarchitecture and application models. Possible improvements to KitFox framework include the following items.

- Optimization of KitFox processes to reduce multi-physics simulation overhead
- Support of more libraries such as power delivery network (PDN) models
- Updating the integrated models to the latest versions
- Addition of system or metric-driven optimization modules

CHAPTER IV

CHARACTERIZATION AND MANAGEMENT OF PERFORMANCE AND LIFETIME RELIABILITY TRADEOFF

Continued device miniaturization and compact integration of transistors in a chip raise lifetime reliability concerns for future multicore processors. Traditionally processors were designed with large guard bands to guarantee the worst-case operations. However, in practice applications rarely operate at the limits, and the advance of microarchitectural adaptations such as power or thermal management enables processors to adapt executions and avoid operating at extreme conditions. In physically constrained processors, adding large design margins is a costly solution and prohibits performance growth. Therefore, microarchitectural approaches such as dynamic reliability management (DRM) have gained favor as cost-efficient solutions to enhance processor lifetime reliability. In this research, reliability problems refer to degradation phenomena and related lifetime issues, although processor reliability includes various classes of issues such as soft errors or electrostatic discharge.

This chapter presents an approach to tackling lifetime reliability challenges in multicore processors by bridging the gap between the physics of device operations and application behaviors. In a multicore processor, cores experience different levels of stresses depending on application characteristics, power states, and thermal behaviors. As a result, the processor produces uneven degradation across the multicore die, and the cores that experience stronger stresses become vulnerable to earlier failures. Processor-level lifetime and throughput are eventually limited by these early failing components.

In addition, there exists a fundamental tradeoff between performance and lifetime reliability. High-performance operations generate more switching activities of microarchitecture components, which are accompanied by increased power and heat dissipations that

accelerate aging phenomena. On the other hand, enhancing lifetime reliability favors low utilization to reduce stresses and thus chance of failures. Therefore, DRM is not merely about enhancing processor lifetime but must balance the tradeoff between performance and reliability. The challenges are 1) characterizing how the execution of parallel applications creates degradation variations on a multicore die, 2) understanding how such variations affect processor-level lifetime and performance, 3) quantifying the lifetime reliability and performance tradeoff, and 4) using these understandings to develop adaptive techniques that manage the tradeoff between processor reliability and performance. Towards these, this research makes the following contributions:

- *Lifetime reliability modeling in cycle-level microarchitecture simulations*:

  This work incorporates lifetime reliability models into cycle-level microarchitecture simulations with transient power and thermal modeling. Reliability characteristics are characterized with respect to spatiotemporally varying voltage and thermal states that are induced by workload dynamics and adaptive controls.

- *Characterization of workload-induced degradation distribution on a multicore die*:

  This research characterizes workload-induced degradation distributions in a multicore processor. It shows that the degradation distribution caused by parallel executions can be characterized by using a probabilistic model such as normal distribution.

- *Variance-aware reliability management*:

  Based on the preceding characterization, this research shows that the variance of degradation distribution is a critical factor to determine processor lifetime. Thus, controlling processor executions to reduce the variance of the distribution effectively leads to processor-level lifetime enhancement.

- *Quantification of performance and reliability tradeoff*:

  This dissertation points out an important fact that performance and lifetime reliability are inversely related. This study introduces a new metric to quantify this tradeoff and

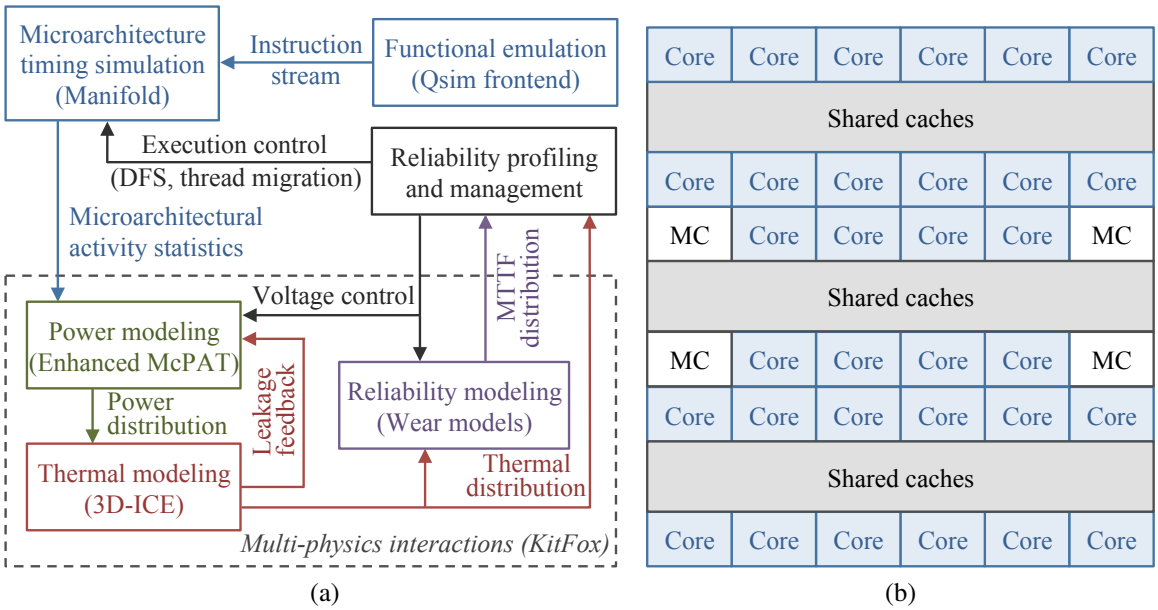


Figure 18: (a) Full-system cycle-level microarchitecture simulation with coordinated multi-physics interactions using KitFox. (b) Homogeneous processor floorplanning with 32 out-of-order cores.

shows that the metric is maximized when both performance and reliability are balanced rather than performing throughput or reliability-oriented operations.

- *Managing performance and reliability tradeoff*:

  It is demonstrated that the performance-reliability tradeoff can be enhanced via variance-aware reliability management using adaptation techniques such as thread migration, dynamic voltage scaling, or turbo-mode execution.

4.1 Experiment Method

This analysis is based on full-system microarchitecture and multi-physics simulations comprised of an application functional emulator, microarchitecture timing simulator, and coordinated multi-physics framework as illustrated in Figure 18(a). This section describes the simulation setup and reviews how KitFox framework presented in Chapter 3 is used for architecture-level lifetime reliability modeling via the integrated microarchitecture and multi-physics simulations.

Table 4: Simulation Setup for Architecture-Level Lifetime Reliability Modeling of Multi-core Processors

Models	Description
Frontend	Qsim (QEMU) x86 functional emulation [35]
Benchmarks	Multi-threaded PARSEC & SPLASH-2 benchmark suites [7]
Cores	32 out-of-order cores (timing model), each core with 128-entry re-order buffer, 6-issue width, and 80-entry load-store queue
Caches	32KB coherent L1 and 1MB shared L2 caches per core [6]
On-chip network	6×6 torus network
Memory system	4 MCs, 1 channel, 2 ranks, 8 banks
Power model	McPAT [40] is substantially enhanced to support DVFS and leakage power updates. 16nm technology models are used.
Thermal model	3D-ICE [70] is configured to simulate a 2D package with transient thermal modeling.
Reliability model	Wear models in Table 2 are adjusted to meet 5 years of processor MTTF at baseline conditions (defined at $T = 65°C$, $V = 0.8V$).
Multi-physics interactions	KitFox framework orchestrates interactions between power, thermal, reliability, and other runtime variables (e.g, voltage, clock frequency) with Manifold microarchitecture timing simulator.

4.1.1 Full-System Microarchitecture and Multi-Physics Simulation Environment

Manifold microarchitecture timing simulator [81] is configured to model a homogeneous processor comprised of 32 out-of-order cores. Figure 18(b) shows the floorplan of simulated 32-core processor. Cores with coherent cache hierarchy [6] are connected in a 6×6 torus network. Qsim frontend emulator (QEMU-based) [35] boots a Linux kernel and executes x86 parallel application binaries to drive the cycle-level microarchitecture timing models. PARSEC [8] and SPLASH-2 [84] benchmarks are used in the experiment. In each simulation, a benchmark is fast-forwarded to the region of interest to skip initialization phases, and then timing simulation is performed until the benchmark finishes. Benchmarks are executed in multi-threaded mode using all 32 cores of the processor. Table 4 summarizes the simulation setup.

KitFox framework presented in Chapter 3 is used to coordinate interactions among multiple physical models including McPAT [40], 3D-ICE [70], and RAMP-like failure models

[72, 71, 74]. In particular, McPAT is substantially enhanced to support transient power modeling with DVFS and leakage power-temperature feedback. 3D-ICE is configured to simulate a 2D package with a conventional air cooling model. Transient thermal modeling is used in 3D-ICE. RAMP-like failure models [72, 71, 74] in Table 2 and Section 3.2.3 are used for lifetime reliability modeling. The failure models include hot carrier injection, electro-migration, negative-bias temperature instability, stress migration, and time-dependent dielectric breakdown. These models are adjusted to meet 5 years of processor-level MTTF at baseline conditions defined at $T = 65^{\circ}$C and $V = 0.8$V. This is referred to as *baseline MTTF* in experiment discussions.

In the full-system microarchitecture and multi-physics simulations using KitFox, multiple physical models are simultaneously simulated with the microarchitecture timing simulator as shown in Figure 18(a). Execution of application binaries through the frontend functional emulator feeds the microarchitecture timing models with instructions to simulate. The microarchitecture simulator collects the timing information and activity counters of functional components that are used to estimate the power dissipation of modeled components. Power results are mapped onto core-level floorplans shown in Figure 18(b), and thermal field is calculated at package level based on the input power distributions. Temperature changes cause feedback interactions with leakage power. Cumulative failure rates are calculated at the floorplan blocks with respect to time-varying operating conditions including voltage and temperature states. The chain of these physical interactions creates a loop and is repeated during the transient simulation. Based on the collected reliability profiles (i.e., core failure rates and lifetime prediction), adaptive controls are applied to regulate degradation states using thread migration, dynamic voltage scaling, or turbo-mode execution.

4.1.2 Architecture-Level Lifetime Reliability Modeling

With billions of transistors in a chip and continuously increasing device density at every technology node, processor-level lifetime reliability modeling becomes a statistical analysis. Since the failure mechanisms reflect long-term behaviors, they are impractical to simulate across the processor with device-level details. In this study, architecture-level multi-physics modeling is used to calculate time-varying failure rates and evaluate lifetime reliability. As discussed with Eq. (2) and (3) in Section 3.2.3, the cumulative failure rates of cores are calculated with respect to time-varying stress conditions including voltage and temperature states that are induced by workload dynamics and microarchitectural operations. Resulting TTF of cores are estimated based on the calculated failure rates. Instead of using one representative value (e.g., average temperature) for the entire processor or relying on greatly simplified microarchitectural models [27, 28, 34, 56, 71, 72, 73, 74, 85, 87], the multi-physics simulation framework implemented by KitFox in this research uses interacting physical models during microarchitecture simulations for lifetime reliability characterization. As discussed in Section 3.2.3, common exponential models are used to express the failure rate λ of different failure mechanisms listed in Table2. Although this approach may not give precise prediction of unknown future operations, the likelihood estimation of TTF can be used to address the relative reliability criticality of different applications or adaptive controls in multicore processors.

4.2 Lifetime Reliability Characterization of Multicore Processors

This section presents a characterization of workload-induced degradation distributions on a multicore die and discusses how these distributions affect processor-level lifetime reliability based on full-system microarchitecture and multi-physics simulations.

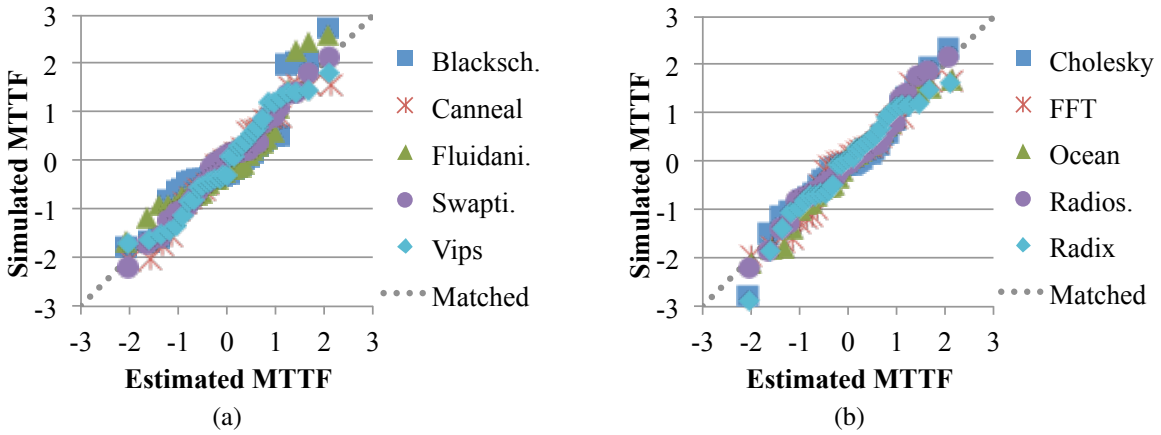


Figure 19: Comparison between simulated TTF and estimated TTF distributions based normal distribution modeling for (a) PARSEC and (b) SPLASH-2 benchmarks. TTF distributions are scaled to standard normal.

4.2.1 Characterizing Spatial Distribution of Degradation

Non-uniform degradation in a multicore processor leads to variations in core-level lifetime. The TTF of the multicore processor depends on how many failed cores would be tolerated until the processor is regarded as inoperable. To define the failure of the processor, *operability threshold* [62] is defined as Eq. (4). The processor is regarded as failed if there are less number of operable cores remaining on the multicore die than the operability threshold. Therefore, the processor TTF is determined as a function of the operability threshold.

$$\text{Operability threshold} = \frac{\text{Minimum \# of operable cores}}{\text{Total \# of cores}} \tag{4}$$

Based on the experimental results of full-system microarchitecture and multi-physics simulations, core-level TTF distributions on the multicore die are characterized. At the end of simulation of each benchmark, mean (μ) and variance (σ^2) are calculated from the samples of core TTF distribution on the multicore die. For an unidentified distribution, a normal-like distribution is assumed. Random samples are generated from the normal distribution with the same mean and variance of benchmark simulation results. Figure 19 shows that the generated normal distribution samples closely match the simulation results. In this figure, TTF distributions are scaled to the standard normal of zero mean and unit standard

Table 5: Reliability Characterization: Normal Distribution Models of PARSEC and SPLASH-2 Benchmarks

PARSEC	$N(\mu,\sigma)$	SPLASH-2	$N(\mu,\sigma)$
Blackscholes	$N(1.285,0.097)$	Cholesky	$N(1.499,0.112)$
Canneal	$N(2.264,0.019)$	FFT	$N(2.224,0.019)$
Fluidanimate	$N(1.398,0.121)$	Ocean-nc	$N(2.309,0.014)$
Swaptions	$N(1.824,0.060)$	Radiosity	$N(1.682,0.098)$
Vips	$N(2.056,0.032)$	Radix	$N(2.178,0.026)$

deviation. Figure 19(a) compares the results of PARSEC benchmarks, and the subplot (b) shows the comparison of SPLASH-2 benchmarks. This reveals that core TTF distributions on the multicore die due to the parallel execution of applications can be characterized by using a normal distribution model. Table 5 lists the reliability characteristics of PARSEC and SPLASH-2 benchmarks based on the normal distribution assumption. Mean and standard deviation in the table are normalized to the baseline MTTF, where $\mu = 1.0$ means the baseline TTF. This observation brings a new insight of characterizing lifetime reliability distributions in multicore processors. Notably, mean and variance are the generic characteristics of any random distributions, so this approach can be applied to other distribution models as well.

The observation of normal distribution enlightens how core TTF distributions will appear when parallel applications are multiplexed without execution controls over a long period of time. Assume that an application i creates normally distributed TTF variation $N(\mu_i,\sigma_i)$ on a multicore die, and the normal distributions created by n different applications are independent. It is also assumed that the execution time of each application is identical such that the normal distributions are independent and identically distributed (i.i.d.). For the i.i.d. normal distributions created by application $i = 0,1,2,...,n-1$, the resulting core TTF distribution $N(\mu,\sigma)$ is expressed as Eq. (5), where the mean μ and standard deviation

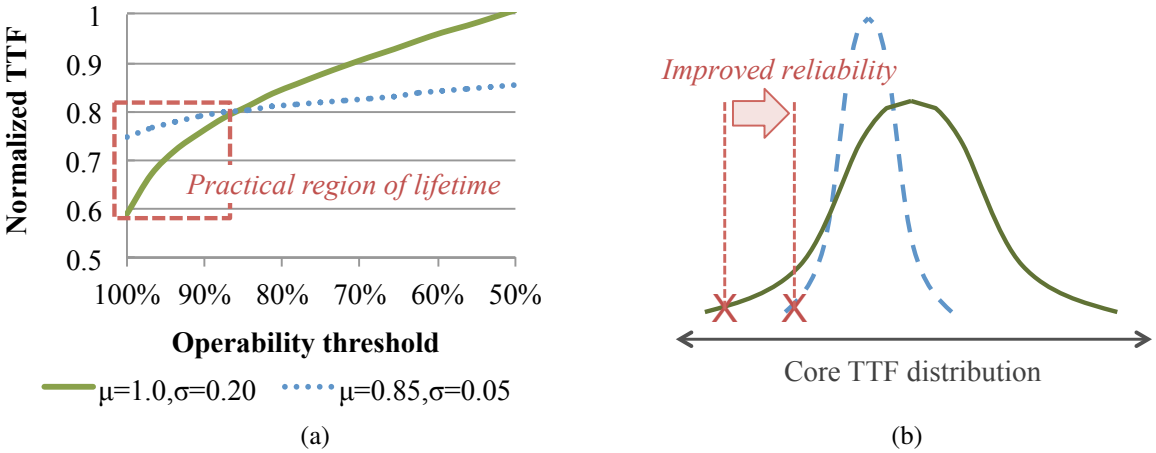


Figure 20: (a) In the practical region of lifetime, the distribution with smaller variance provides better lifetime even with lower mean. (b) Reshaping the TTF distribution by reducing the variance improves processor lifetime.

σ are the average of μ_i and σ_i for all i, respectively.

$$N(\mu,\sigma) = N\left(\frac{1}{n}\sum_{i=0}^{n-1}\mu_i\ ,\ \ \frac{1}{n}\sqrt{\sum_{i=0}^{n-1}\sigma_i^2}\right) \tag{5}$$

Eq. (5) reveals that the degradation variation on the multicore die will ideally fade away when parallel applications are simply multiplexed over time for $n \to \infty$. However, in practice time-multiplexed normal distributions are not perfectly i.i.d. since initial states depend on wear and thermal distributions created by previous applications. In addition, system-level tasks such as thermal or power management (e.g., dynamic voltage scaling or turbo-mode executions) do not necessarily generate normally distributed degradation on the multicore die. Therefore, variance-aware reliability management is still necessary, and the following section discusses how such variations affect processor-level lifetime reliability based on the normal distribution assumption.

4.2.2 Effect of Degradation Variance on Processor Lifetime

Based on the observation of normally distributed degradation on the multicore die, it is analyzed how the variance of core-level TTF distribution affects processor lifetime reliability. Figure 20(a) shows the TTF changes of two different cases; 1) high TTF mean and large

variance and 2) low TTF mean and small variance. The first case represents a situation that the processor has low average degradation (and therefore high TTF mean) with large non-uniformity in degradation distribution on the multicore die. The second case is that the processor has more degradation on average (and thus low TTF mean) but with relatively even degradation across cores. It is possible for the processor to tolerate a few failures to extend lifetime by sacrificing peak performance, but it is impractical to use the processor with many failures. The most practical use cases will be tolerating few core failures (e.g., within 10% failing or above 90% operability threshold), which is denoted by *the practical region of lifetime* [62] in Figure 20(a). In this region, avoiding early failures effectively leads to processor lifetime enhancement. Thus, this graph shows that reducing the variance of core TTF distribution is a key to improving processor lifetime. This concept is referred to as *dynamic reliability variation management* (DRVM) [67] and illustrated in Figure 20(b). This figure shows that reshaping the core TTF distribution by reducing the variance delays early failures despite the smaller mean of the distribution. This is distinct from typical wear-leveling approaches that refer to evening out activities in cores via microarchitectural heuristics [16, 18, 27].

4.3 Quantification of Performance and Lifetime Reliability Tradeoff

There exists a fundamental tradeoff between performance and lifetime reliability. High-performance operations generate more switching activities of microarchitecture components, which are accompanied by increased power and heat dissipations that accelerate degradation processes. On the other hand, lifetime reliability favors lower utilization to reduce stresses and resulting failure rates. Therefore, dynamic reliability management techniques cannot simply work to improve processor lifetime but must balance the tradeoff between performance and reliability.

Processor throughput and lifetime are inversely related as shown in Figure 21(a). Each square mark in this graph is plotted based on the simulation results shown in Table 5, and a

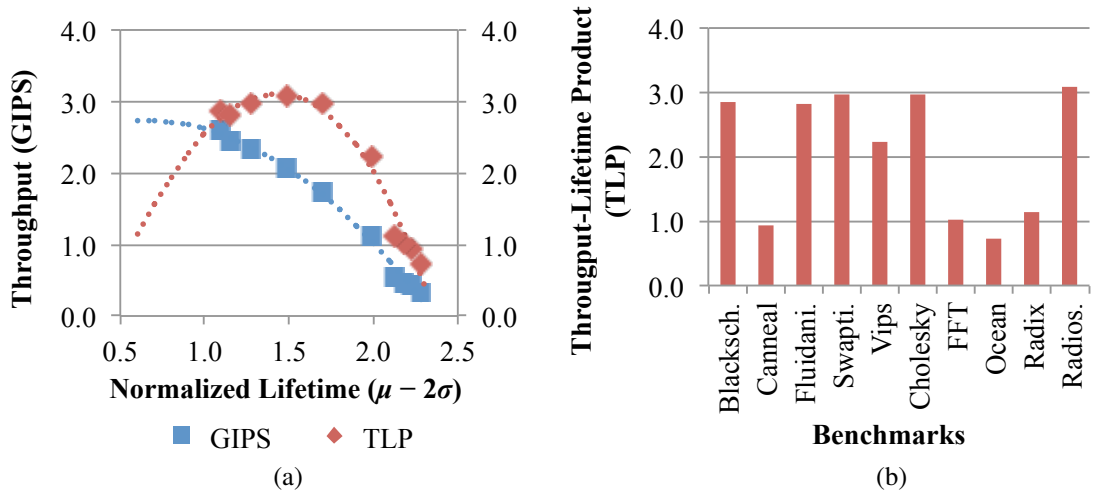


Figure 21: (a) Inverse relation between performance and lifetime reliability. (b) Throughput-lifetime product evaluation of simulated benchmarks.

trend line of these sample points is drawn. For each simulation of a benchmark, instruction counts and execution time are measured, and average Giga instructions per second (GIPS) is calculated to represent processor throughput. The x-axis of the graph is equivalent to processor lifetime that is calculated as $(\mu - 2\sigma)$, where μ and σ are obtained from the simulation results in Table 5. With the normal distribution assumption, $(\mu - 2\sigma)$ corresponds to approximately 97% operability threshold, which is translated that the 32-core processor would tolerate about one core failure. The selection of operability threshold (e.g., 2σ) is a design choice.

When the processor produces high throughput (e.g., compute-bound workloads), there are more switching activities of microarchitecture components, which increase power and heat dissipations. Consequently, the execution of such workloads has an adverse impact on processor reliability. Lifetime reliability may be naively improved by regulating processor operations and power consumptions, but it is traded with performance loss. To evaluate this tradeoff, a new metric, *throughput-lifetime product* (TLP), is introduced [67]. The TLP metric is expressed as in Eq. (6).

$$\text{TLP} = \text{GIPS} \times (\mu - 2\sigma) \tag{6}$$

In this equation, the performance term is represented with the average throughput (GIPS) of the processor, and the reliability term is expressed by $(\mu - 2\sigma)$ of core TTF distribution. A reason for using this metric can be easily understood with an analogy of energy-delay product (EDP) that quantifies a tradeoff between performance and energy. Similarly, the TLP metric shows the *reliability efficiency* of the processor. Larger TLP means that the processor produces more throughput for the amount of degradation occurring during executions. On the contrary, low TLP indicates that the processor execution is not efficient and exhibits relatively more degradation for the generated throughput.

In Figure 21(a), the TLP metric shows a parabolic trend as a function of lifetime or throughput. Because of the inverse relation between performance and lifetime reliability, high-throughput operations exhibit worse processor lifetime, thereby yielding low TLP. At the other end of the curve, low-performance operations put minimal stresses on the processor and thus show longer expected lifetime, but it is also poorly evaluated with the TLP metric because of low throughput. For example, continued execution of the Ocean benchmark will result in greater processor lifetime than executing other benchmarks according to the results in Table 5. However, when evaluated with the TLP, it is the worst benchmark in that it produces less throughput for the amount of degradation occurring in the execution. Therefore, the TLP metric provides a method to quantify and evaluate the performance and reliability tradeoff in multicore processors.

4.4 Microarchitectural Adaptations to Manage Performance and Lifetime Reliability Tradeoff

This section describes microarchitectural adaptation techniques based on the notion of DRVM (i.e., variance-aware reliability management) to improve the performance-reliability tradeoff using the TLP metric. The presented DRVM techniques are 1) phase-aware thread migration (PATM), 2) dynamic voltage scaling (DVS) to reshape core TTF distributions, and 3) turbo mode execution (TME) combined with DVS-based variance management. Although these techniques appeared in various different implementations for dynamic power,

thermal, or reliability management [15, 16, 18, 34, 45, 46, 74], the contribution of this research is that they are grounded in the fundamentals of degradation variations on a multicore die to manage processor performance and lifetime reliability tradeoff. It is assumed that degradation monitoring is readily available in the processor such as using aging sensors. In this study, core-level TTF is used as a proxy for degradation information. The implementations of microarchitectural adaptation techniques are first described, and then results are discussed.

4.4.1 Phase-Aware Thread Migration (PATM)

Thread migration is typically known as a dynamic thermal management (DTM) technique that is used to spread out thermal distributions and avoid creating hotspots. It was also suggested as a DRM method in previous work [16, 18] since temperature has strong correlation with lifetime reliability. However, dynamic reliability management fundamentally differs from DTM in that control decisions based on instantaneous properties such as instructions per second (IPC) or temperature readings [15, 16] do not necessarily reflect the *cumulative* behaviors of aging phenomena. For a similar reason, simply relying on degradation monitoring (e.g., aging sensors) also causes inefficiency in the reliability management. For instance, if a thread being executed on a hot core enters a low-throughput (e.g., memory-bound) or idle phase, it is better to keep this thread in the same core rather than moving it to another core whose thread may have transitioned to a high-power state and thus will exacerbate the problem if thread swapping occurs between these two cores. Therefore, a thread migration method has to utilize both instantaneous performance metrics and cumulative degradation profiles to effectively manage lifetime reliability and reshape core TTF distribution on the multicore die.

In the phase-aware thread migration, the cumulative failure rates of cores are measured at every monitoring interval to account for spatiotemporally varying thermal states. Since

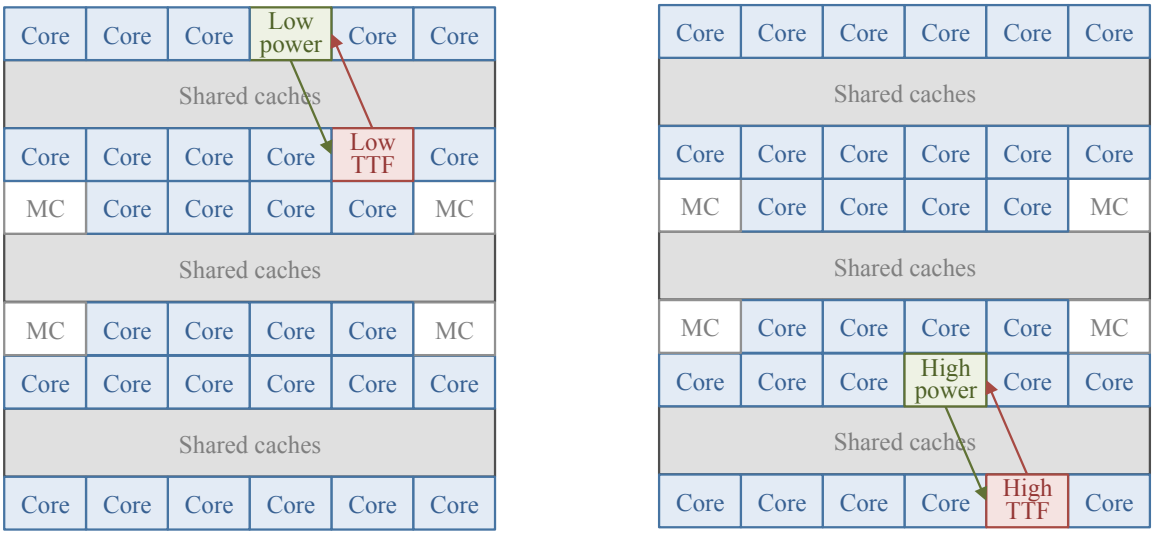


Figure 22: Coordinated thread swapping (a) between cores with low estimated TTF and power and (b) between cores with high estimated TTF and power for dynamic reliability variance management.

the voltage and clock frequency of cores are unchanged in this operation mode, degradation variations are primarily caused by non-uniform thermal states across the cores. It is assumed that degradation monitoring is available on the die such as using aging sensors, and this study does not discuss the details of degradation detection mechanisms. The TTF of each core is predicted from calculated failure rates, and μ and σ of core TTF distribution are measured.

Since the goal of DRVM is to reshape core TTF distribution by reducing the distribution variance, thread migration is triggered when the variance of the distribution exceeds a predefined threshold set to $\sigma_{th} = 0.05$ in the experiment. When the processor exhibits relatively even degradation across cores (i.e., $\sigma \ll \sigma_{th}$), employing thread migration has minor improvements to processor lifetime. Thread migration method is limited in its ability to minimize σ (e.g., $\sigma \to 0$) or precisely control the failure rate of cores since the effectiveness of this technique is subject to the power and thermal behaviors of individual threads being migrated across cores.

PATM also monitors the IPC of threads being executed on separate cores. Low-IPC

operations generate less switching activities of microarchitecture components and thus dissipate less power and heat. These low-IPC threads are relocated to weak cores to slow down their degradation processes. When the PATM is invoked ($\sigma > \sigma_{th}$), it swaps threads between cores that have the lowest IPC and TTF. Similarly, another set of thread swapping is made between cores that have the largest IPC and TTF to avoid μ of the core TTF distribution being biased towards high-TTF cores. Such coordinated thread swapping illustrated in Figure 22 is applied to the next pair of cores if their expected TTF also deviate more than σ_{th} from the mean of the distribution. After migrating the threads, these cores are protected from being invoked for another thread migration to avoid the threads being continuously tossed around the cores. The timeout also allows the cores to have enough time to recover from deviated degradation status. The weakness of using thread migration technique for DRVM is that degradation adjustment is not precisely controllable but strongly depends on the behaviors of migrating threads.

4.4.2 Dynamic Voltage Scaling (DVS)

A DVS method exploits voltage impact on reliability as shown by the models in Table 2 to adjust the failure rate of individual cores and eventually reduce the variance of core TTF distribution. Voltage scaling techniques were widely studied in previous work for power or thermal regulations, and a few studies attempted to exploit it for dynamic reliability management [15, 16, 34, 45, 46]. The prior studies applied dynamic voltage scaling to adjust the failure rate of individual cores executing independent threads (e.g., multi-programmed executions) to meet a reliability target. However, they did not consider the consequence of DVS on core TTF distribution and processor-level lifetime when executing parallel applications (e.g., multi-threaded executions). Dynamic voltage scaling for DRVM differs from these prior studies in that it is applied to reshape core TTF distribution by reducing the variance (σ^2) instead of regulating the mean (μ) of the distribution such that performance impact is minimized. Regulating the mean of core TTF distribution affects the

Table 6: Voltage Scaling Table for Dynamic Reliability Variance Management

$\mu_{core} - \mu$	Voltage	$\mu_{core} - \mu$	Voltage
-0.10	0.774V	$+0.10$	0.828V
-0.15	0.762V	$+0.15$	0.843V
-0.20	0.750V	$+0.20$	0.860V
-0.30	0.729V	$+0.30$	0.894V

entire threads of a parallel workload, but reducing the variance only controls a few deviant threads and therefore has a minor impact on overall performance.

The difficulty of utilizing the DVS method is in determining the degree of voltage scaling required to adjust degradation by a desirable amount since failure rate is a function of both voltage and temperature, and they have different impacts on different failure models as shown in Table 2. To simplify the problem, a voltage scaling table such as Table 6 is created with predicted TTF changes at the baseline conditions. The cumulative failure rates of cores are measured at every sampling interval (i.e., 1ms in the experiment), and μ and σ of core TTF distribution are calculated. Distance to the mean of TTF distribution is measured for each core, which is expressed as $\mu_{core} - \mu$ in the voltage scaling table (Table 6). Then, necessary voltage adjustment is applied to each core using this lookup table. Voltage scaling in general has better controllability for DRVM than the thread migration method, but it may have a negative impact on performance because of clock frequency changes accompanied by the voltage scaling.

4.4.3 Turbo-Mode Execution (TME)

An aggressive execution control such as turbo-mode execution boosts core operations by elevating voltage and clock frequency levels to increase throughput. Instead, performance improvements are traded with increased voltage and thermal stresses that accelerate degradation behaviors and thus diminish lifetime reliability. However, if the turbo mode is applied for a relatively short duration compared to the overall execution time, changes in failure rates can be kept small while drawing throughput increase. When turbo-mode execution

is not engaged, voltage scaling-based DRVM (described in Section 4.4.2) is performed as a base operation mode to reshape core TTF distribution and compensate for the reliability penalty due to the turbo execution.

For the TLP metric expressed as in Eq. (6), turbo-mode execution attempts to increase the throughput term (GIPS) but instead sacrifices the mean (μ) of the reliability term because of increased voltage and thermal stresses. It is observed that boosted executions in general amplify the variance (σ^2) as well, which exacerbate the reliability problem by magnifying the non-uniformity of core TTF distribution. Therefore, turbo-mode execution alone is not sufficient to enhance performance-reliability tradeoff that is evaluated with the TLP metric, and a compensatory operation such as DRVM adaptation has to append to offset the reliability penalty.

Turbo-mode execution can be effective if it returns good performance improvement. In a multicore processor, contentions for shared resources prohibit the turbo execution from increasing throughput [42]. Therefore, two metrics are used to determine the initiation of turbo mode in this study; 1) core utilization and 2) cache miss rates. Core utilization is calculated as the ratio of active cycles over total clock cycles during a monitoring interval, and it is subject to how the operating system (OS) schedules the threads of parallel applications across cores. If a core pipeline is kept busy, increasing the clock frequency of the core effectively leads to performance improvement. When core utilization is greater than 99.9%, cache miss rates are used as indicators to trigger the turbo mode. A miss-rate threshold is empirically obtained from simulation results such that it is small enough to forbid memory-bound operations from being involved in the turbo mode but also large enough for applications to have enough chances of employing turbo executions.

When a turbo-mode execution is triggered, the voltage and clock frequency of cores are elevated to a maximum power state that meets a predefined power cap. Since the purpose of this study is to demonstrate how performance-reliability tradeoff can be leveraged via the notion of DRVM and TLP, this study does not focus on developing an optimal turbo

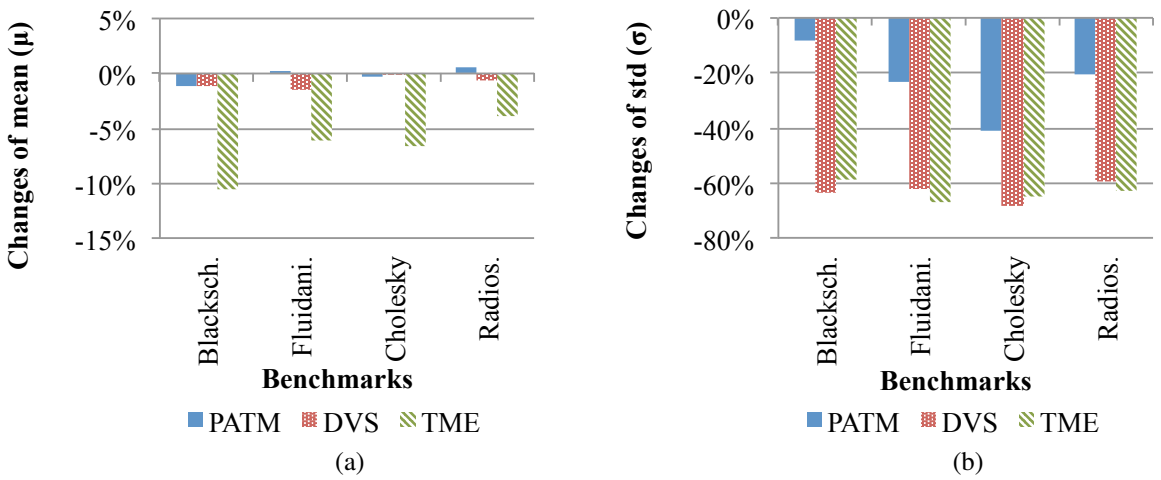


Figure 23: Changes of (a) mean μ and (b) standard deviation σ of core TTF distributions by applying dynamic reliability variance management (DRVM) with phase-aware thread migration (PATM), dynamic voltage scaling (DVS), and turbo-mode execution (TME) combined with DVS-based variance control.

execution method, which requires a distinct set of tasks and experiments such as the work of Lo et al. [42]. When turbo mode is not engaged, this technique performs DRVM via dynamic voltage scaling as a base operation to reshape core TTF distribution and therefore mitigate the reliability penalty due the turbo-mode execution.

4.4.4 Evaluation of Performance and Lifetime Reliability Tradeoff

Since a key to dynamic reliability variance management is to reduce the variance of degradation distribution for effective processor lifetime enhancement, four benchmarks that show large non-uniformity in core TTF distributions without execution controls (i.e., $\sigma \gg \sigma_{th} = 0.05$) are selected from the application characterization results shown in Table 5; Blackscholes, Fluidanimate, Cholesky, and Radiosity. Three microarchitectural adaptation techniques described in the previous sections are applied to these benchmarks, and results are discussed. Although these adaptation techniques can be applied to other benchmarks, there are limited improvements since those applications natively show even degradation across cores.

Figure 23 shows changes in the mean and standard deviation of TTF curves by applying

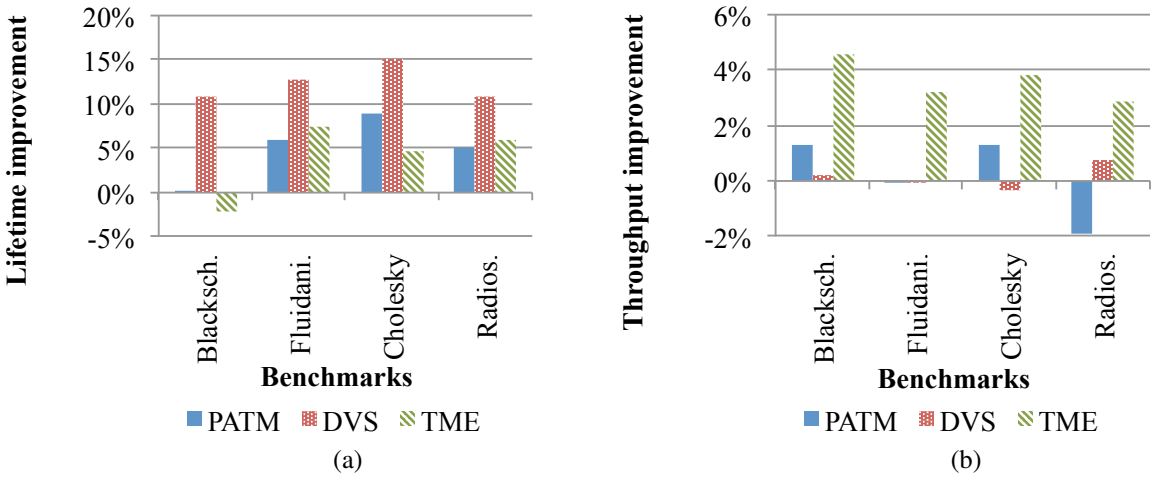


Figure 24: Changes in (a) Lifetime and (b) throughput by applying DRVM techniques.

the adaptation techniques compared to uncontrolled executions. Both phase-aware thread migration and dynamic voltage scaling for DRVM attempt to reshape core TTF distribution by reducing the variance (σ^2) but does not control the mean (μ). Results in Figure 23 show that both adaptation techniques have minor changes to the mean but greatly reduce the standard deviation of core TTF distribution, which effectively leads to improved processor lifetime as plotted in Figure 24(a). For instance, the voltage scaling method applied to the Fluidanimate benchmark unintentionally decreases the mean of core TTF distribution by 2% (normalized to the baseline MTTF), but 62% reduction in standard deviation compensates for the shift of the mean. Collectively, it results in 13% improvement in processor lifetime.

Results in Figure 24(a) show that the voltage scaling technique produces greater lifetime improvement than the tread migration method. The effectiveness of thread migration is limited by the behaviors of migrating threads. The PATM works for applications that have distinct power states across threads, but it does not perform well with those with similar power dissipations across cores such as Blackscholes. Migrating the threads of similar power states does not effectively re-distribute thermal stresses and therefore shows limited lifetime improvements over the uncontrolled executions.

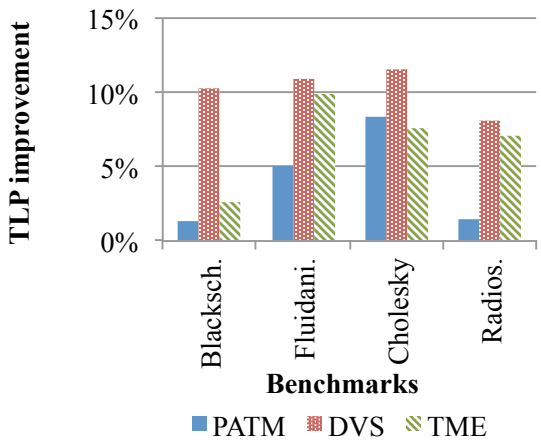


Figure 25: Improvement of throughput-lifetime product (TLP) by DRVM techniques.

In overall, both phase-aware thread migration and dynamic voltage scaling for DRVM improve the performance and reliability tradeoff that is evaluated with the TLP metric as shown in Figure 25. These techniques also show minor impact on throughput while achieving processor lifetime enhancement, since they do not modify the mean of core TTF distribution that has strong correlation with the overall performance of the processor. Therefore, Figure 25 shows that the TLP improvements of these two techniques have similar trends as the changes of lifetime shown in Figure 24(a).

Turbo-mode execution attempts to increase throughput by sacrificing the mean (μ), but the decreased lifetime due to the shift of mean is compensated by DRVM adaptation that reduces the variance. As shown in Figure 23(a), turbo-mode execution significantly diminishes the mean of core TTF distribution because of accelerated degradation. Since dynamic reliability variance management using voltage scaling has minor impact on the mean of core TTF distribution, the changes are primarily due to the turbo-mode execution. Instead, the DRVM adaptation via voltage scaling significantly reduces the variance of core TTF distribution as shown in Figure 23(b). As a result, the turbo-mode execution combined with DRVM has minor impact on overall processor lifetime, while achieving performance improvement. For example, a compute-bound workload such as Blackscholes operates

Table 7: Impact of Dynamic Reliability Variance Management Techniques on Controlling Throughput, Lifetime, and TLP Metric

DRVM type	Throughput	Lifetime ($\mu - 2\sigma$)		TLP
		μ	σ	
PATM	-	-	$\downarrow$	$\uparrow$
DVS	-	-	$\Downarrow$	$\Uparrow$
TME + DVS	$\Uparrow$	$\Downarrow$	$\Downarrow$	$\uparrow$

longer in turbo mode since it can draw more performance increase from boosted executions than other benchmarks. Applying the turbo-mode execution to such a benchmark causes more degradation and decreases processor lifetime, which is traded with throughput improvement as shown in Figure 24. Consequently, the turbo-mode execution combined with DRVM also enhances performance and reliability tradeoff that is measured by the TLP as shown in Figure 25. However, it is observed that the reliability penalty of turbo-mode execution is greater than the throughput benefit. Therefore, the TLP improvement of this operation mode is less than that of DVS-based variance management without employing turbo executions.

Table 7 summarizes the impact of three microarchitectural adaptation techniques on processor throughput, lifetime, and TLP. Phase-aware thread migration method improves the TLP metric by reducing σ of core TTF distribution, but the improvement is limited since the effectiveness of this method is bounded by the behaviors of migrating threads. Dynamic voltage scaling technique achieves the most increase in TLP by effectively regulating σ of core TTF distribution. Turbo-mode execution trades throughput improvement with decreased μ. Hence, the turbo execution alone cannot enhance the throughput-lifetime product, but a compensatory operation is required to mitigate the reliability penalty caused by the turbo mode. This can be achieved by appending DVS-based variance management when the turbo-mode execution is not employed. In overall, combined TME + DVS operations also improve performance and lifetime reliability tradeoff in the multicore processor, but it does not achieve as large TLP improvements as DVS-based variance management

without turbo-mode executions since the reliability penalty is greater than the performance benefits.

4.5 Summary

Microarchitectural approaches such as dynamic reliability management have gained favor as cost-efficient solutions to enhance processor lifetime reliability. However, without understanding the basic physics of device operations and application behaviors, dynamic reliability management has to rely on microarchitectural heuristics. This research contributes to characterizing how the parallel execution of applications creates degradation distributions on a multicore die and how such distributions affect processor lifetime. A finding reveals that reducing the variance of degradation distribution is a key to improving processor lifetime. In this research, it is also claimed that dynamic reliability management cannot simply work to improve processor lifetime but must balance the tradeoff between performance and reliability, which can be evaluated by the introduced throughput-lifetime product metric.

CHAPTER V

EXTENDING AMDAHL'S LAW FOR UNDERSTANDING LIFETIME RELIABILITY, PERFORMANCE, AND ENERGY EFFICIENCY OF HETEROGENEOUS PROCESSORS

The paradigm of designing processors is shifting from simply improving performance to enhancing energy (or power) efficiency, as it has become a critical barrier to microarchitectural operations. Heterogeneous multicore processors have been studied as alternative implementations to improve energy efficiency and performance. For instance, a processor comprised of a complex core (i.e., out-of-order execution) and many small cores (i.e., in-order execution), such as the one shown in Figure 26(a), can enhance these metrics by using the big core for faster sequential executions and many simple cores for energy-efficient parallel operations. Such microarchitectural asymmetry is referred to as *heterogeneous* in this research. Energy efficiency and performance improvements of the heterogeneous processor over a conventional homogeneous processor are governed by Amdahl's Law [4] as widely studied in prior work [12, 13, 17, 23, 31, 38, 39, 47, 75, 83].

Extending prior work for the performance and energy (and power) modeling of heterogeneous processors, this research presents the lifetime reliability, performance, and energy efficiency models of heterogeneous multicores and discusses the consequences of such heterogeneous designs. Lifetime reliability behavior of a heterogeneous processor can also be characterized by using Amdahl's Law. Depending on Amdahl's scaling (or parallelization) factor f, processor composition (e.g., number of big and small cores), or thread scheduling method, stresses can be biased to a particular type of core. For instance, assume that the heterogeneous processor includes only one complex core such as Figure 26(a) and also utilizes the complex core during parallel executions to maximize performance increase via

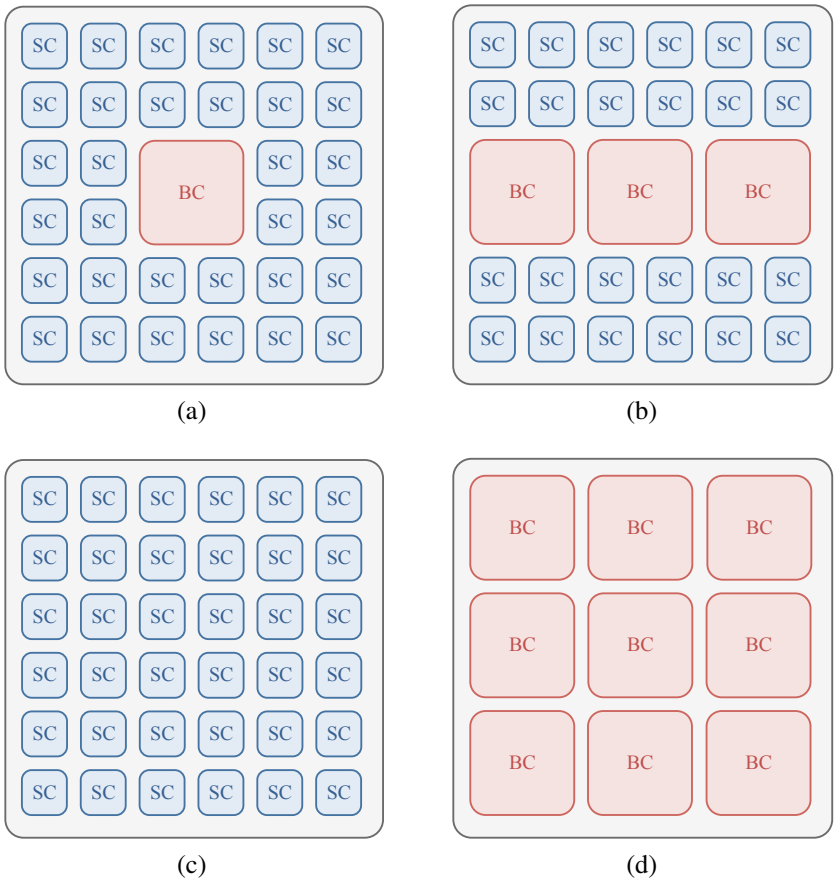


Figure 26: Multicore configurations: (a) heterogeneous processor with one big core (BC) and many small cores (SC), (b) heterogeneous processor with multiple big cores and fewer small cores, (c) homogeneous processor of small cores, and (d) homogeneous processor comprised of big cores.

techniques such as bias scheduling [38] or accelerating critical threads [31, 75]. The big core is the busiest computing unit in that it is always turned on and has to execute both serial and a part of parallel phases of a workload. It is subject to extended stresses compared to other computing units, and such *biased stresses* become worse when the application spends more time on sequential operations. This is generally not a critical issue in homogeneous multicore processors, since any one of the cores can be selected to execute sequential operations. Load-balancing, wear-leveling, or proposed DRVM (described in Chapter 4) methods can be applied to the homogeneous cores to even out degradation [18, 27, 62, 67]. On the other hand, if sequential executions are substantially short, the chance of failure is

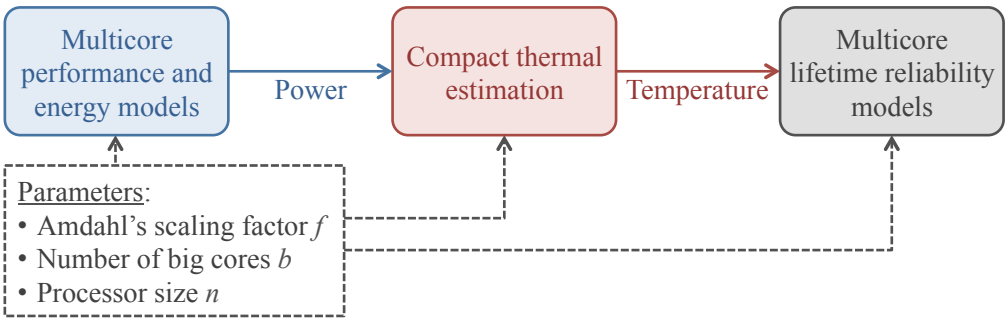


Figure 27: Modeling flow of performance, energy efficiency, thermal, and lifetime reliability characterization of heterogeneous multicore processors.

greater among many simple cores. When a small core fails, the heterogeneous processor may tolerate the failure if graceful degradation is allowed for a number of duplicated small cores on the die [18, 27, 73]. However, the failure of the only big core immediately leads to the failure of the entire processor since no other cores can replace the role of the failed big core without serious performance degradation.

If the heterogeneous processor accommodates a few number of big cores such as Figure 26(b), any one of them can be selected to perform serial executions. Unused cores can be power-gated to minimize degradation and save power. Consequently, the failure rate of complex cores can be greatly reduced by sharing loads. However, increasing the number of big cores on the die reduces the small core count under the same area constraint. It may decrease the peak throughput of parallel executions, especially for small-size processors where relatively large portion of the area would be taken by big cores.

Alternatively, different scheduling methods can be considered to mitigate the reliability problem of the heterogeneous processor. For instance, a complex core can be used only to execute sequential operations and turned off during parallel phases. This type of scheduling policy is particularly advocated in power-constrained processors where not all cores might be able to turn on because of power limitation [17, 83]. Such a scheduling method also limits the peak throughput of parallel executions, but the power-gated big core benefits from alleviated stresses and improves overall reliability. Therefore, the lifetime reliability of heterogeneous multicore processors strongly depends on processor configuration (e.g.,

number of big and small cores) and utilization (e.g., scheduling methods), characterized by Amdahl's Law. This research proposes and uses the approach shown in Figure 27 to evaluate the performance, energy efficiency, thermal, and lifetime reliability of multicore processors. As shown, this research makes the following contributions.

- *Performance and energy efficiency models*:

  This research extends heterogeneous multicore models presented in the work of Hill and Marty [23] and Woo and Lee [83] to include multiple complex cores and utilize power-gating (of unused cores) in performance and energy calculations.

- *Thermal estimation for lifetime reliability modeling*:

  A compact thermal model is proposed to estimate the temperature of hypothetical heterogeneous processors with different processor compositions (i.e., number of big and small cores) and execution phases (i.e., serial or parallel executions) for accurate lifetime reliability modeling.

- *Lifetime reliability models of heterogeneous multicore processors*:

  Using the preceding models, this research presents the lifetime reliability models of heterogeneous processors and shows that multicore reliability can be characterized by using Amdahl's Law.

- *Assessing the performance, energy efficiency, and lifetime reliability of heterogeneous multicore processors*:

  This research analyzes the performance, energy efficiency, and lifetime reliability consequences of heterogeneous processors, which are traded as a function of processor size n, big core count b, and Amdahl's scaling factor f.

5.1 Revisiting Amdahl's Law for Performance and Energy Scaling of Multicore Processors

This research adopts the performance and energy models from previous work [23, 83] and extends them to analyze the lifetime reliability of heterogeneous processors. This section reviews the performance and energy models of various multicore configurations with updated assumptions.

5.1.1 Homogeneous Processor of Simple (Small) Cores

It is assumed that a baseline homogeneous processor is comprised of n number of small cores, following the modeling methodology presented in the work of Hill and Marty [23]. Figure 26(c) illustrates the homogeneous multicore processor composed of small cores. According to Amdahl's Law, maximum performance speed-up is given as Eq. (7). Performance is improved by parallelizing the f fraction of computations with n cores [23]. This is an optimistic performance estimation without thread parallelization or migration overhead. The f fraction in the equation is referred to as *Amdahl's scaling factor* or *parallelization fraction* in this research.

$$Perf_{hom:s} = \frac{1}{(1-f)+\frac{f}{n}} \tag{7}$$

An energy model is adopted from the work of Woo and Lee [83], but the assumption is modified such that idle cores (e.g., unused $n-1$ number of cores during serial executions) are ideally turned off and do not contribute to processor power dissipation. It is assumed that the power consumption of a simple core is normalized to 1. Hence, processor power dissipation during serial executions is equivalent to single-core power, and the processor consumes $n \times 1$ amount of power when all cores are active to execute parallel threads. With the normalization, energy scaling based on Amdahl's Law becomes $E_{hom:s} = 1$, and power scales the same as the performance model in Eq. (7), $W_{hom:s} = Perf_{hom:s}$.

5.1.2 Homogeneous Processor of Complex (Big) Cores

In a homogeneous processor composed of big cores such as Figure 26(d), it is assumed that each big core has s times better performance and r times larger area than those of a small core [13, 23, 83]. Pollack's Rule [49] states that performance and area are correlated as $s \propto \sqrt{r}$. Within the same total area as the homogeneous processor of small cores, there can be up to n/r number of big cores. The performance speed-up of the homogeneous processor of complex cores is calculated as Eq. (8). The improvement is achieved by accelerating serial executions (i.e., $1-f$ fragment of a workload) by an s times faster big core and parallelizing the f fraction of the load by n/r number of big cores.

$$Perf_{hom:b} = \frac{1}{\frac{1-f}{s} + \frac{f}{s} \times \frac{r}{n}} \tag{8}$$

Another parameter p is considered to represent the relative power of a big core, meaning that the big core consumes p times more power than a small core. This power expression is adopted from the work of Chung et al. [13], where power and area (or performance) are correlated as $p \propto (\sqrt{r})^{\alpha}$ and α is set to 1.75. The energy dissipation of the processor is expressed as Eq. (9). In this equation, it is also assumed that unused cores are power-gated. During serial executions, the processor consumes p amount of power that is equivalent to single big-core power. It dissipates $p \times (n/r)$ amount of power when all big cores are active for parallel executions.

$$E_{hom:b} = \frac{1-f}{s} p + \frac{f}{s} \times \frac{r}{n} \times \frac{n}{r} p \tag{9}$$

5.1.3 Heterogeneous Processor with Maximum Scheduling

Departing from a simple heterogeneous model that has only one big core and many small cores as studied in prior work [13, 17, 23, 83], the heterogeneous configuration is generalized to incorporate multiple big cores. When executing only one application at a time, one complex core is sufficient to handle the serial part of the application. However, in a

general situation such as multiplexed applications and system operations (e.g., virtual environment), there can be a need for including multiple complex cores to handle concurrent serial executions of multiple workloads. Therefore, it is a valid design for the heterogeneous processor to include multiple big cores, and such a design is considered in this analysis. *Maximum scheduling* is assumed for the heterogeneous processor such that it can fully utilize computing units and maximize performance.

$$Perf_{het:ms} = \frac{1}{\frac{1-f}{s} + \frac{f}{b \times s + (n - b \times r)}} \tag{10}$$

It is assumed that the heterogeneous processor is comprised of b number of big cores, and the rest of the area is populated with $n - b \times r$ small cores. The total area is equivalent to those of homogeneous processors. A selected complex core is used to execute sequential operations, and parallel executions make use of all cores in the processor to maximize performance. The performance speed-up of the heterogeneous processor with multiple big cores and maximum scheduling is expressed as Eq. (10). Single-thread executions ($1 - f$ part of a workload) are accelerated by a complex core that has s times greater performance than a simple core. The f fraction of the workload is parallelized by both big cores ($b \times s$ speed-up) and small cores ($n - b \times r$ speed-up).

$$E_{het:ms} = \frac{1-f}{s} p + \frac{b \times p + (n - b \times r)}{b \times s + (n - b \times r)} f \tag{11}$$

During sequential executions, the processor consumes p amount of power that is equivalent to single big-core power. Other unused cores are assumed to be ideally turned off and do not contribute to the power dissipation. In parallel phases, the total power is the sum of $b \times p$ by big cores and $(n - b \times r) \times 1$ by small cores, where the power dissipation of a small core is normalized to 1 and a big core is assumed to have p times larger power than that of a small core. Collectively, the total energy of the heterogeneous processor with multiple big cores and maximum scheduling is calculated as Eq. (11).

5.1.4 Heterogeneous Processor with Dynamic Scheduling

Another possible case of utilizing the heterogeneous processor is separating the use of distinct core types. For instance, complex cores are only used to execute serial threads, and parallel operations are run only on simple cores. This type of *dynamic scheduling* is advocated especially in power-constrained processors, where big cores may not be able to run simultaneously with a group of small cores because of power limitation [17, 83]. Running parallel threads only on simple cores also solves the scheduling issues caused by performance imbalance between different core types. The performance speed-up of the heterogeneous processor with multiple big cores and dynamic scheduling is calculated as Eq. (12). The sequential part $(1-f)$ is accelerated by a big core that is s times faster than a small core, and the f fraction is parallelized by $n-b\times r$ number of small cores.

$$Perf_{het:ds} = \frac{1}{\frac{1-f}{s} + \frac{f}{n-b\times r}} \tag{12}$$

The total energy of the heterogeneous processor with dynamic scheduling is expressed as Eq. (13). The processor selects a big core to perform single-thread executions and consumes power p. Other cores are assumed to be power-gated or used by other applications, where the power dissipation of those cores attribute to other applications. In parallel phases, threads are run only on small cores and consume $n-b\times r$ amount of power.

$$E_{het:ds} = \frac{1-f}{s}p + \frac{n-b\times r}{n-b\times r}f \tag{13}$$

5.1.5 Composed Processor of Simple (Small) Cores

Hill and Marty presented a hypothetical homogeneous processor model comprised of n small cores, where a set of cores are dynamically combined and help each other to speed up serial executions such as thread-level speculation or helper threads [23, 48]. It is assumed that these helper threads run on separate cores, and a set of r small cores have the same performance as one complex core denoted by s. Instead, the group of small cores consumes

Table 8: Comparison of Simple and Complex-Core Homogeneous Processor Pairs

	IBM Blue Gene/Q	IBM POWER7	Intel Atom Z520	Intel i7 960
Core execution type	In-order	Out-of-order	In-order	Out-of-order
Technology node	45nm	45nm	45nm	45nm
Estimated die area	360mm^2	567mm^2	26mm^2	263mm^2
Number of cores	18	8	1	4
Cores-to-die area ratio	34%	32%	37%	37%

Table 9: Area Scaling Factors Compared to Previous Generation Technologies

	IBM POWER7+	Intel i7 2700K	Intel i7 3770K
Core execution type	Out-of-order	Out-of-order	Out-of-order
Technology node	32nm	32nm	22nm
Core area scaling from prev. gen.	0.68×	0.66×	0.66×
Number of cores	8	4	4
Cores-to-die area ratio	37%	37%	37%

r amount of power that can be greater than the power dissipation of a big core p. The performance speed-up of the *composed processor* is expressed as Eq. (14), and the total energy is calculated as Eq. (15).

$$Perf_{com} = \frac{1}{\frac{1-f}{s} + \frac{f}{n}} \tag{14}$$

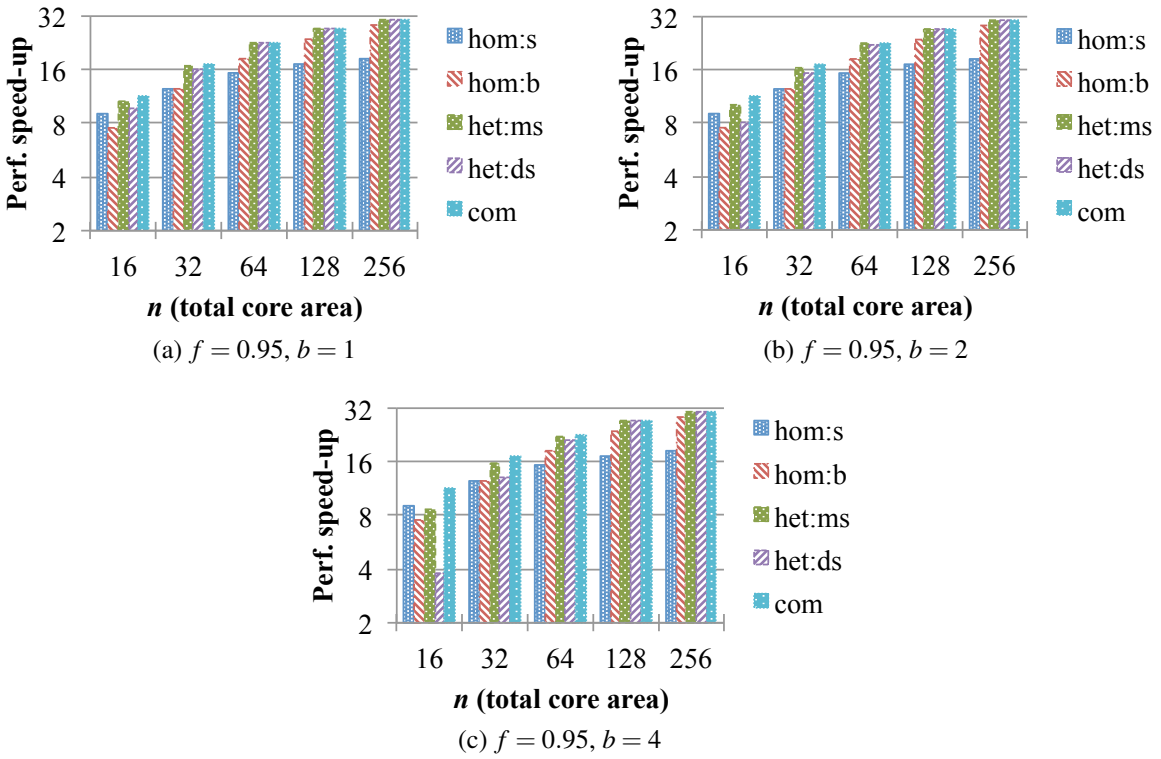
The composed processor accelerates serial executions $(1-f)$ by s times, and the f part is parallelized by n small cores. This processor represents an ideal case in that it can accelerate both serial and parallelizable parts of workloads. Since r number of cores are grouped to yield s times greater performance, the processor dissipates $r\times 1$ amount of power during serial executions and $n\times 1$ in parallel phases.

$$E_{com} = \frac{1-f}{s} r + \frac{f}{n} n \tag{15}$$

5.2 Evaluating Performance and Energy Scaling of Multicore Models

This section evaluates the multicore performance and energy models presented in the previous section to correlate their impact with lifetime reliability. An out-of-order core in general has 2-4× larger area than a comparable in-order design. Table 8 summarizes the area of complex and simple core pairs from IBM and Intel processors, estimated from available references and die shots [13, 22, 33, 79, 90]. The area ratio between big and small cores is estimated around 2.5-4.4× for these processors. Based on these examples, an integer number $r = 3$ is chosen as the area ratio to compose hypothetical heterogeneous processors in this research. Pollack's Rule [49] states that performance and area are correlated as $s \propto \sqrt{r}$. A power expression is adopted from Chung's model [13], where the power is expressed as $W \propto \sqrt{r}^{\alpha}$ and $\alpha = 1.75$. These parameters are applied to Eq. (7)-(15) for the performance and energy efficiency evaluation of multicore processors. Table 8 shows that the proportion of core area on the die is relatively consistent, ranging between 30-40% of the die. The processors at successive technology nodes in Table 9 also show similar cores-to-die area ratio. No discernible correlation is found among these cases between the area ratio and core types, number of cores, or other uncore configurations (e.g., cache sizes, on-chip network). Hence, this analysis focuses on the core scaling factors and simplifies other conditions.

Table 9 shows that core size scales by 0.66-0.68× every technology node. With continued scaling, it is predicted that there can be around a hundred simple cores within the die area similar to Intel i7 at 8nm technology node, or about twice more on a much larger IBM POWER7 die. In this analysis, core count n (in unit of small cores) is scaled between $n = 16$ and 256 as shown in Figure 28. This figure plots the maximum performance speed-up of multicore processors with varying n and fixed $f = 0.95$. The big core count of the heterogeneous models differs in each sub-plot. When $n \gg 64$, it is observed that the overall performance speed-up is limited by sequential throughput as parallel executions become substantially short (when maximum performance increase is assumed with increasing



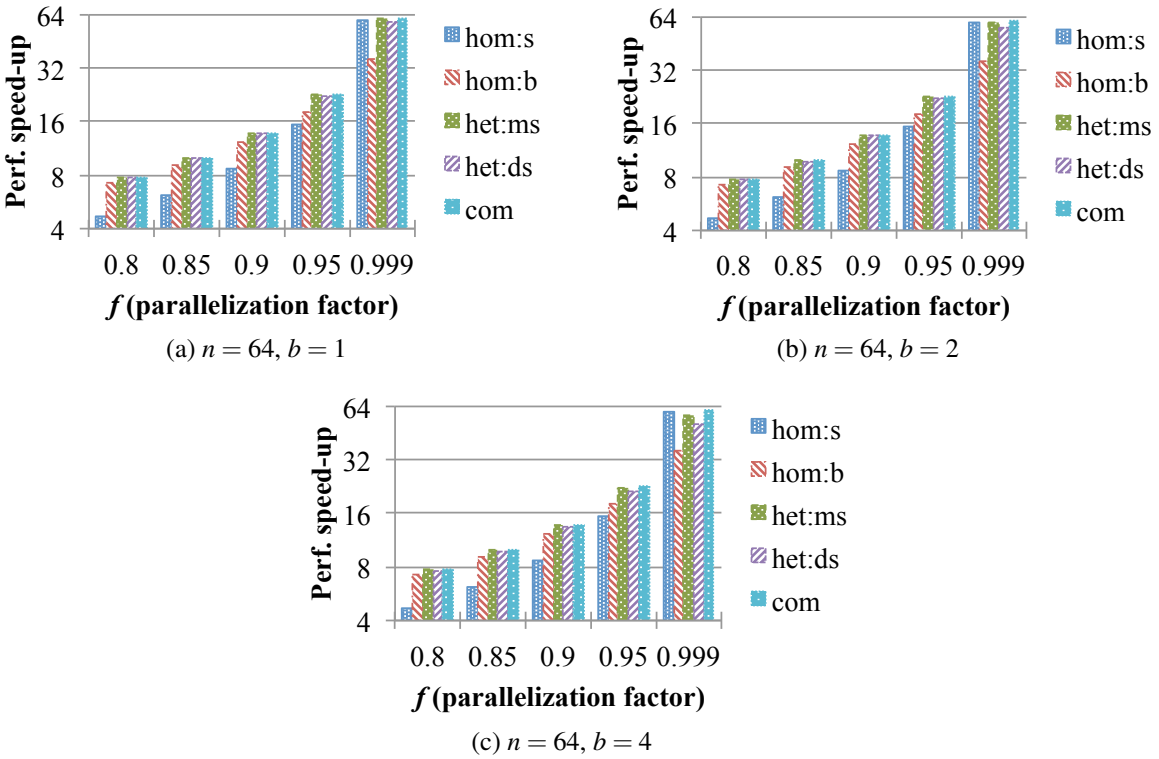
(a) $f = 0.95, b = 1$ (b) $f = 0.95, b = 2$

(c) $f = 0.95, b = 4$

Figure 28: Maximum performance speed-up of multicore processors with parallelization factor $f = 0.95$, and varying total area (n in unit of small cores) and number of big cores (b) in the heterogeneous processors.

n). Hence, the homogeneous processor composed of simple cores suffers from low performance. Little performance difference is made in heterogeneous processors by varying the number of big cores (b) when n is large. On the other hand, the cases with $n \ll 64$ are more dominated by parallel performance because of narrow parallelization width; relatively longer time is spent on parallel executions. The heterogeneous processor of small n with dynamic scheduling and multiple big cores as in Figure 28(c) shows limited performance increase since this processor utilizes only simple cores for parallelization. Based on these observations, an intermediate size of $n = 64$ is selected as an exemplary case to study.

Figure 29 shows the performance speed-up of various multicore configurations with $n = 64$ and Amdahl's scaling factor between $f = 0.8$ and 0.999. The number of complex



(a) $n = 64$, $b = 1$

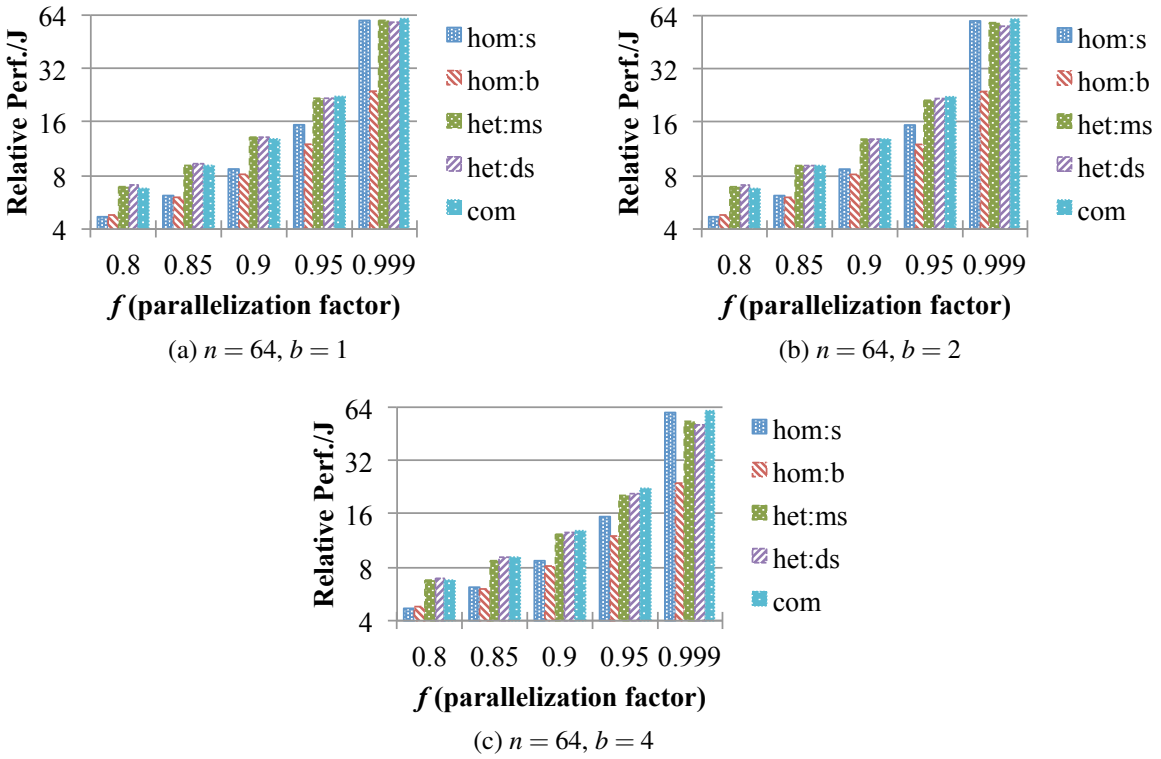
(b) $n = 64$, $b = 2$

(c) $n = 64$, $b = 4$

Figure 29: Maximum performance speed-up of multicore processors with $n = 64$ and parallelization fraction scaled between $f = 0.8$ and 0.999. The number of big cores (b) in the heterogeneous processor is varied in the sub-plots.

cores in the heterogeneous designs is varied by $b = 1$, 2, and 4 in the sub-plots. The composed processor as an ideal design provides the most performance speed-up, and the heterogeneous processors also produce similar performance increases. For moderately parallelizable workloads (e.g., $f = 0.8$), sequential operations are important in that good amount of time is spent on performing single-thread executions. In this case, the homogeneous processor made of complex cores shows as good performance speed-up as the heterogeneous or composed processor.

As parallelization fraction f increases, the overall performance speed-up is dominated by parallel executions. At $f = 0.999$, the homogeneous processor of big cores produces only about 60% throughput of other multicore designs, whereas the other homogeneous option with small cores delivers as much performance speed-up as the composed processor. If the number of complex cores is increased in the heterogeneous processors (e.g., $b = 1$



(a) $n = 64$, $b = 1$

(b) $n = 64$, $b = 2$

(c) $n = 64$, $b = 4$

Figure 30: Relative energy efficiency (performance per Joule) of multicore processors for $n = 64$ with scaling factor between $f = 0.8$ and 0.999 and different number of big cores (b) in the heterogeneous processors.

to 4), peak parallel throughput decreases particularly when $f \to 1.0$. However, increasing big core count has a minor impact if f is small. From the comparison of Figure 28(a) and (c), big core count shows greater impact for small n ($\ll 64$) since complex cores occupy relatively large area in small-size processors, and the opposite happens for large n. Thus, the effect of increasing the number of big cores is more addressed for large $f \to 1.0$ or small n.

When comparing the subplots (a) and (b) of Figure 28 and 29, there are subtle differences in performance for the heterogeneous processors by increasing the number of big cores from $b = 1$ to 2. However, increasing the big core count in heterogeneous processors makes a large difference in lifetime reliability, and this is discussed later in this chapter. Further increasing the number of complex cores such as $b = 4$ penalizes the heterogeneous processors especially when handling highly parallel workloads (e.g., $f = 0.999$).

Energy efficiency can be calculated by using Eq. (7)-(15). Following the methodology presented in the work of Woo and Lee [83], performance per Joule (*Perf./J*) is used to represent the energy efficiency. Figure 30 plots the relative energy efficiency of multicore processors under the same conditions as Figure 29. In overall, the results show similar trends as those in Figure 29. The composed processor provides the most increase in energy efficiency as $f \rightarrow 1.0$. The heterogeneous processors produce similar or even better energy efficiency when the parallelization fraction f is small since it is assumed that a group of dynamically combined small cores would dissipate more power than a single complex core despite the same performance (refer to Section 5.1.5). The homogeneous processor of big cores suffers from low energy efficiency. This processor spends relatively longer time on executing the parallelizable fraction f that activates all cores. For a similar reason, increasing the number of big cores in the heterogeneous processors reduces the peak parallel throughput when $f \rightarrow 1.0$ such as Figure 30(c). As a result, the heterogeneous processors achieve less improvement in energy efficiency when $b = 4$ and $f = 0.999$ as shown in Figure 30(c) compared to the composed processor.

5.3 Compact Thermal Estimation

Reliability characteristics strongly depend on temperature. Therefore, assuming a constant core failure rate is not a useful approach, and this section describes a method to estimate the thermal states of multicore processors for accurate lifetime reliability modeling.

Using the first-order ordinary differential equation (ODE) [21, 89], temperature is modeled as Eq. (16). $\mathbf{x}$ is the temperature vector of n equal-sized blocks (e.g., small cores as floorplans) at time i ($i = 0$ is an initial state), and $\mathbf{A}$ matrix shows heat spreading process. Matrix $\mathbf{B}$ includes conversion factors from power input $\mathbf{u}$ to temperature $\mathbf{x}$, where an entry of the matrix B_{jk} denotes the conversion rate of power dissipation at location j to thermal increase at location k. The vector $\mathbf{w}$ is the effect due to ambient temperature.

$$\mathbf{x}(i) = \mathbf{A}^i\mathbf{x}(0) + (\mathbf{A}-\mathbf{I})^{-1}(\mathbf{A}^i-\mathbf{I})\mathbf{B}\mathbf{u} + \mathbf{w} \tag{16}$$

In this analysis, thermal effects are estimated by a steady-state model since lifetime reliability is governed by long-term behaviors. Steady-state temperature is denoted by the vector $\mathbf{x}$ when $i \to \infty$ as shown in Eq. (17). $\mathbf{x}'$ means $\mathbf{x}(\infty)$, and $\mathbf{A}^i \to 0$ in Eq. (16) for $i \to \infty$.

$$\mathbf{x}' = (\mathbf{A} - \mathbf{I})^{-1}\mathbf{B}\mathbf{u} + \mathbf{w} \tag{17}$$

Assume that vector $\mathbf{x}'$ is the temperature vector of the homogeneous processor of small cores with 100% parallel executions, where all cores are active and each small core has the normalized power of 1 (refer to Section 5.1.1). This thermal state vector $\mathbf{x}'$ is referred to as *baseline state*, and it is analyzed how different processor compositions or execution phases (e.g., single-thread executions) create thermal differences to the baseline. The purpose of compact thermal modeling is to estimate temperature difference to the baseline state rather than accurately calculating the absolute magnitude of temperature. By substituting a portion of the small-core die with complex cores, it creates changes to the power distribution that is expressed as $\Delta\mathbf{u}$. Power and thermal changes are also created within a processor depending on execution modes (e.g., serial or parallel phases). In any scenarios, changes in power distribution ($\Delta\mathbf{u}$) result in the temperature difference of $\Delta\mathbf{x}'$ as shown in Eq. (18). $(\mathbf{A} - \mathbf{I})^{-1}\mathbf{B}$ is substituted with a matrix $\mathbf{C}$ in the equation.

$$\Delta\mathbf{x}' = (\mathbf{A} - \mathbf{I})^{-1}\mathbf{B}\Delta\mathbf{u} = \mathbf{C}\Delta\mathbf{u} \tag{18}$$

The entries of matrix $\mathbf{C}$ are the steady-state power-to-thermal conversion factors between any two locations on the die. For instance, thermal change at the block j is expressed as $\Delta x'_j$ that is the sum of two terms as shown in Eq. (19). The first term in this equation, $C_{jj}\Delta u_j$, means temperature change due to the power difference (with respect to the baseline) at the same location. The second term, $\sum C_{kj}\Delta u_k$, is the thermal contribution to location j as a result of power changes at other locations $k \neq j$.

$$\Delta x'_j = C_{jj}\Delta u_j + \sum_{k \neq j}^{n} C_{kj}\Delta u_k \tag{19}$$

The matrix **C** depends not only on the thermal properties of package but also floorplanning of multicore die. For a hypothetical heterogeneous processor without known floorplanning (or many possible combinations of floorplanning), thermal behaviors are estimated by using a compact scalar model as in Eq. (20). In this equation, the thermal change of block j is primarily induced by the power difference at the same location (Δu_j) multiplied by conversion factor C_{jj}. Thermal impact by other blocks to location j is estimated by calculating the average of power change $\Delta\bar{u}$ of other blocks multiplied by scaling factor $\bar{C}_{kj}$. These scaling factors can be obtained from thermal models (e.g., HotSpot [29]) by varying power input at location j and measuring thermal changes at location j for C_{jj} and $k \neq j$ for $\bar{C}_{kj}$, where $\bar{C}_{kj}$ is the average of C_{kj} for all k.

$$\Delta x'_j = C_{jj}\Delta u_j + \bar{C}_{kj}\Delta\bar{u} \tag{20}$$

When block j belongs to a big core, $\Delta\bar{u}_b$ is used as shown in Eq. (21), and $\Delta\bar{u}_s$ is used for a small core in Eq. (22) to represent $\Delta\bar{u}$ of Eq. (20). Using these equations, it becomes possible to estimate thermal differences between heterogeneous and baseline homogeneous processors, as well as thermal changes between execution phases within a processor.

$$\Delta\bar{u}_b = \frac{(b-1)r}{n-r}\delta(p_b-1) + \frac{n-b\times r}{n-r}\delta(p_s-1) \tag{21}$$

$$\Delta\bar{u}_s = \frac{b\times r}{n-1}\delta(p_b-1) + \frac{n-1-b\times r}{n-1}\delta(p_s-1) \tag{22}$$

In Eq. (21), $\Delta\bar{u}_b$ means the power contribution of all other cores (at location $k \neq j$) to the temperature of a big core that spans over location j. These other cores include $b-1$ number of big cores and $n-b\times r$ small cores. In the first term of Eq. (21), $(b-1)r/(n-r)$ is the area fraction of big cores over the total core area except one big core at location j. When $b=1$, it means that there are no other big cores that affect the temperature of the only big core in the heterogeneous processor. In the area of other big cores expressed as $(b-1)r$, it has the power density difference of p_b-1 compared to that of a small core. p_b is the relative power of a big core, which is $p_b = p/r$ when the core is active or $p_b = 0$ when

Table 10: Validation of Compact Thermal Estimation Compared to HotSpot Steady-State Temperature Model

Processor type	Maximum difference (°C) to a HotSpot model			
	Sequential		Parallel	
	Big core	Small core	Big core	Small core
Homogeneous: small cores	N/A	−0.41	N/A	Baseline
Homogeneous: big cores	+0.63	N/A	+0.45	N/A
Heterogeneous: maximum scheduling	+0.63	Unused	−0.19	−0.01
Heterogeneous: dynamic scheduling	+0.63	Unused	Unused	−0.58
Composed: small cores	N/A	−0.32	N/A	Same as baseline

power-gated. δ is the power density of a small core (in W/cm^2) of the baseline state. In the second term of $\Delta\bar{u}_b$, $(n-b\times r)/(n-r)$ is the area fraction of small cores over the total core area (not including the area of a big core at location j). p_s is the normalized power of a small core that is $p_s = 1$ at active state or $p_s = 0$ when turned off.

Similarly, $\Delta\bar{u}_s$ in Eq. (22) is the power contribution of all other cores to the temperature of a small core at location j. In the first term of Eq. (22), $(b\times r)/(n-1)$ is the area fraction of b big cores over the total core area except one small core at location j, where the area of a small core is normalized to 1. In the area of big cores expressed as $b\times r$, it has the power density difference of $p_b - 1$ compared to that of a small core. p_b is the relative power of a big core, which is $p_b = p/r$ at active state or $p_b = 0$ if turned off. δ is the power density of a small core of the baseline state. In the second term of $\Delta\bar{u}_s$, $(n-1-b\times r)/(n-1)$ is the area fraction of small cores except the one at location j over the total core area. p_s is the normalized power of a small core that is $p_s = 1$ when the core is active or $p_s = 0$ when power-gated.

Table 10 shows the accuracy of the compact thermal estimation compared to a HotSpot steady-state model [29] after calibration. Homogeneous and heterogeneous floorplans are

created under the area constraint of $n = 64$. In the heterogeneous processor, complex cores are placed at the center, similar to Figure 26(a). The multicore processors in HotSpot simulations show as large as 20°C temperature variations between execution phases or core types. Hence, disregarding thermal effects will lead to significant inaccuracy in lifetime reliability modeling. In overall, the compact thermal estimation yields less than 1°C difference to a detailed model, and it greatly simplifies thermal analysis for the lifetime reliability modeling of heterogeneous multicore processors. The estimated temperatures are applied to Eq. (23) and (24) to calculate resulting MTTF.

5.4 Extending Amdahl's Law for Lifetime Reliability Scaling of Multicore Processors

The lifetime reliability of multicore processors is subject to Amdahl's scaling factor f, processor composition with b and n (e.g., number of big and small cores), and scheduling policy (e.g., maximum or dynamic scheduling). This section presents the lifetime reliability models of multicore processors as functions of aforementioned parameters.

5.4.1 Failure Phenomena and Models

Gradual device degradation leads to the failure of processor components. HCI and NBTI are known to be critical failure mechanisms as device technology continues to scale. These failures are primarily caused by charges trapped in the gate oxide that result in the shift of threshold voltage and timing errors [37, 74]. HCI and NBTI models are adopted from the work of Kim et al. [37] and White and Bernstein [82]. Table 11 summarizes the reliability models used in this study.

5.4.2 Modeling of Lifetime Reliability

In this analysis, exponential distribution is used to simplify the models [71, 74, 82]. The failure rate of exponential distribution is expressed as $\lambda = 1/\text{MTTF}$. The total failure rate is calculated as the SOFR of wear mechanisms; $\lambda = \lambda_{HCI} + \lambda_{NBTI}$. It is assumed that the

Table 11: Failure Models Used for Lifetime Reliability Modeling of Multicore Processors

Failure types	Description and models
Hot carrier injection (HCI)	Particles that gain sufficient kinetic energy overcome the barrier to gate oxide and cause degradation [37, 82]. $$\text{MTTF}_{HCI} = A_{HCI}\, I_{sub}^{-n}\, e^{(E_{a\,HCI}/kT)} \quad (23)$$ A_{HCI} = technology-dependent constant, I_{sub} = substrate current, n = acceleration factor, $E_{a\,HCI}$ = activation energy, T = absolute temperature, k = Boltzmann's constant
Negative bias temperature instability (NBTI)	PMOS devices under the negative gate voltage at elevated temperature cause the increase of threshold voltage and timing errors [37, 82]. $$\text{MTTF}_{NBTI} = A_{NBTI}\, V_{gs}^{-r}\, e^{(E_{a\,NBTI}/kT)} \quad (24)$$ A_{NBTI} = process-related constant, V_{gs} = gate voltage, r = voltage acceleration factor, $E_{a\,NBTI}$ = activation energy

failure rate is proportional to the area when stress conditions are identical, and thus the failure rate of a big core (λ_b) is r times greater than that of a small core (λ_s); $\lambda_b = r \times \lambda_s$.

Reliability characteristics strongly depend on temperature as shown in Eq. (23) and (24). The thermal state of a processor differs by core composition (e.g., homogeneous or heterogeneous), execution mode (e.g., serial or parallel), and scheduling policy (e.g., maximum or dynamic scheduling in the heterogeneous processor). This section present how the energy models in Section 5.1 can be translated to thermal states and eventually failure rates of heterogeneous cores across different execution phases.

5.4.3 Homogeneous Processor of Simple (Small) Cores

The reliability state (i.e., failure rates) of a homogeneous processor of simple cores with 100% parallel execution is used as a *baseline* in this analysis. It is assumed that each simple core has the normalized power of 1 and failure rate of λ_s. The total failure rate of

the processor is calculated as Eq. (25), and MTTF is expressed as $1/\lambda_{hom:s}$.

$$\lambda_{hom:s} = (1-f)\frac{\lambda_{s:seq}}{n} + \frac{f}{n} \times n \times \lambda_{s:par} \tag{25}$$

Any small cores on the die can be selected to execute the serial part $(1-f)$ of a workload. The long-term reliability impact of the core executing sequential operations is divided by the number of cores n. It is assumed that unused cores are power-gated and have no increase of failure rates in the mean time. $\lambda_{s:seq}$ is the failure rate of a small core in serial phases at lower operating temperature. Using Eq. (20), it is possible to estimate the temperature difference of the active core executing a serial thread with respect to the baseline (i.e., 100% parallel execution). The estimated temperature difference ($\Delta x'$) is applied to Eq. (23) and (24) to calculate changes in failure rate and resulting MTTF.

During parallel executions, performance improvement (f/n) is offset by correspondingly larger total failure rate $(n \times \lambda_{s:par})$ according to the SOFR. $\lambda_{s:par}$ is the failure rate of a small core of the baseline state, which is normalized to 1 in this analysis.

5.4.4 Homogeneous Processor of Complex (Big) Cores

In a homogeneous processor consisting of complex cores under the same area as the one of simple cores, there can be n/r number of big cores. Applying $b = n/r$ to Eq. (21), $\Delta\bar{u}_b$ becomes $\delta(p_b - 1)$. When $p/r < 1$, it results in lower power density than a simple core and hence reduces the failure rate per unit area because of lower operating temperature. The failure rate of a big core is calculated as $\lambda_b = \sum \lambda_j$ for all j belonging to the big core area. For each λ_j, thermal difference to the baseline state is estimated by using Eq. (20). The total failure rate of the homogeneous processor comprised of complex cores is calculated as Eq. (26).

$$\lambda_{hom:b} = \frac{1-f}{s} \times \frac{r}{n} \times \lambda_{b:seq} + \frac{f}{s \times n/r} \times \frac{n}{r} \times \lambda_{b:par} \tag{26}$$

In this equation, the serial part $(1-f)$ can be executed by any big cores. $\lambda_{b:seq}$ is the failure rate of a big core during sequential phases. Since any complex core on the die can

be chosen to handle serial operations, long-term reliability impact is divided by the number of cores (n/r). The performance increase of parallel executions, $f/(s\times n/r)$, is offset by the sum of failure rates of n/r big cores.

5.4.5 Heterogeneous Processor with Maximum Scheduling

Complex cores in a heterogeneous processor with maximum scheduling are the busiest computing units. One of the complex cores has to execute the serial part of an application, and they also participate in executing parallel threads to maximize performance. For the heterogeneous processor with b number of complex cores, the total failure rate is calculated as Eq. (27).

$$\lambda_{het:ms} = \frac{1-f}{s} \times \frac{\lambda_{b:seq}}{b} + \frac{f}{b\times s + (n - b\times r)} \times \left\{ b\times\lambda_{b:par} + (n - b\times r)\lambda_{s:par} \right\} \tag{27}$$

Any one of the complex cores in the heterogeneous processor can execute the serial part $(1-f)$ of a workload. The long-term reliability impact of the big core ($\lambda_{b:seq}$) in sequential phases is reduced b fold. During parallel executions, total failure rate is calculated as $b\times\lambda_{b:par} + (n - b\times r)\lambda_{s:par}$, where the failure rate of each core type is multiplied by the core count of corresponding type. When a processor failure happens, the probability that the fault is due to big cores is calculated as the failure rate of big cores over the total failure rate ($\lambda_{het:ms}$) as shown in Eq. (28). This equation shows that the reliability of big cores becomes more critical for small b or f since more stresses are put on the big cores.

$$Prob_{b:ms} = \frac{\frac{1-f}{s} \times \frac{\lambda_{b:seq}}{b} + \frac{f\times b\times\lambda_{b:par}}{b\times s + (n - b\times r)}}{\lambda_{het:ms}} \tag{28}$$

5.4.6 Heterogeneous Processor with Dynamic Scheduling

In a heterogeneous processor with dynamic scheduling, distinct type of cores are used to handle different phases of applications. By turning off unused cores, this scheduling policy benefits from improved lifetime reliability that is traded with performance penalty. The total failure rate of the processor is expressed as Eq. (29). The first term in this equation

reflects the reliability impact of big cores during sequential operations, and the second term is the failure rate of small cores in parallel phases.

$$\lambda_{het:ds} = \frac{1-f}{s} \times \frac{\lambda_{b:seq}}{b} + \frac{f}{n-b\times r}(n-b\times r)\lambda_{s:par} \tag{29}$$

The probability that a processor failure is caused by big cores is calculated as Eq. (30) that is the failure rate of big cores over the total failure rate ($\lambda_{het:ds}$). Since any one of big cores can be used to execute serial threads and they are turned off during parallel phases, the reliability criticality is significantly reduced by increasing b. In addition, adding b also decreases $\lambda_{s:par}$ of small cores in Eq. (29) since power-gated complex cores help the heterogeneous processor produce better thermal field in parallel phases.

$$Prob_{b:ds} = \frac{\frac{1-f}{s} \times \frac{\lambda_{b:seq}}{b}}{\lambda_{het:ds}} \tag{30}$$

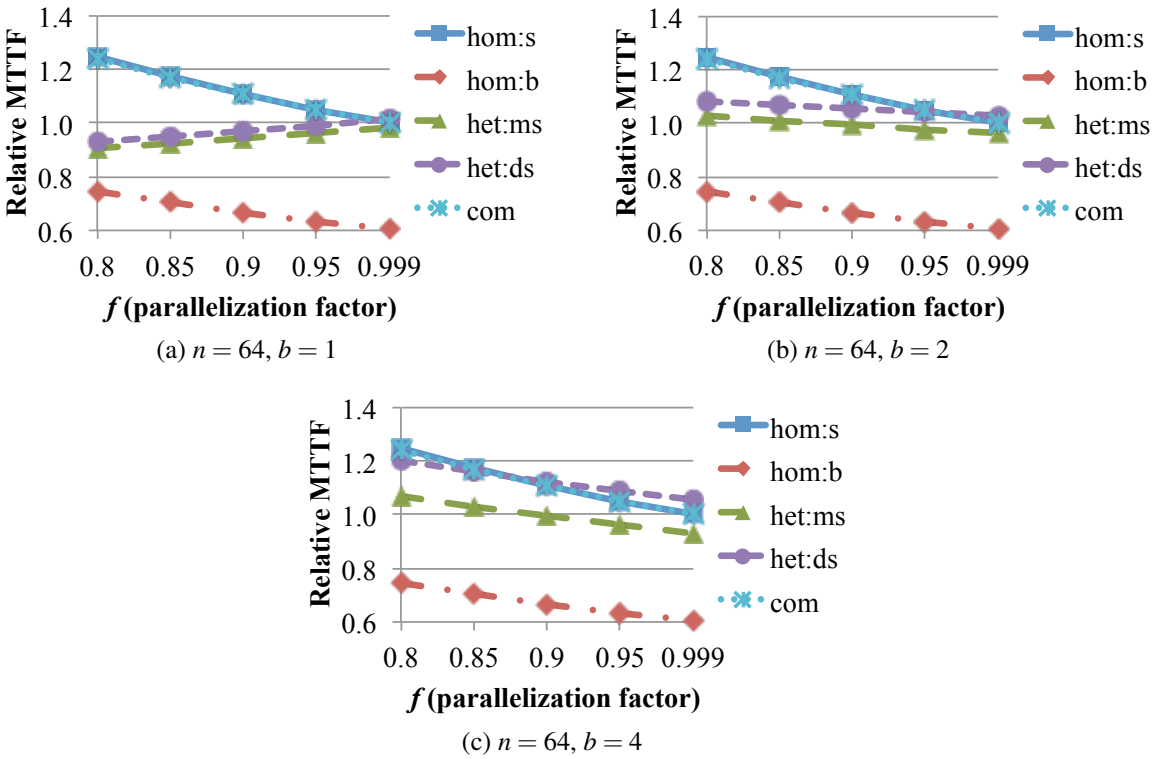
5.4.7 Composed Processor of Small Cores

Composed processor is technically a homogeneous processor comprised of simple cores. The only difference to the conventional homogeneous model is that multiple small cores are grouped to speed up sequential operations. The total failure rate of the processor is shown in Eq. (31). Although this processor utilizes multiple cores in the serial phase of $1-f$, the reliability impact is minor because any r number of small cores among n can be chosen. For parallel executions, the failure rate is calculated in the same way as the homogeneous processor of simple cores (baseline).

$$\lambda_{com} = \frac{1-f}{s} \times \frac{r\times\lambda_{s:seq}}{n} + \frac{f}{n} \times n \times \lambda_{s:par} \tag{31}$$

5.5 Evaluating Lifetime Reliability Scaling of Multicore Models

This section evaluates the lifetime reliability models of multicore processors presented in the previous section. Figure 31 shows the relative MTTF of multicore models ($n = 64$) with Amdahl's scaling factor varied between $f = 0.8$ and 0.999. The number of big cores in the



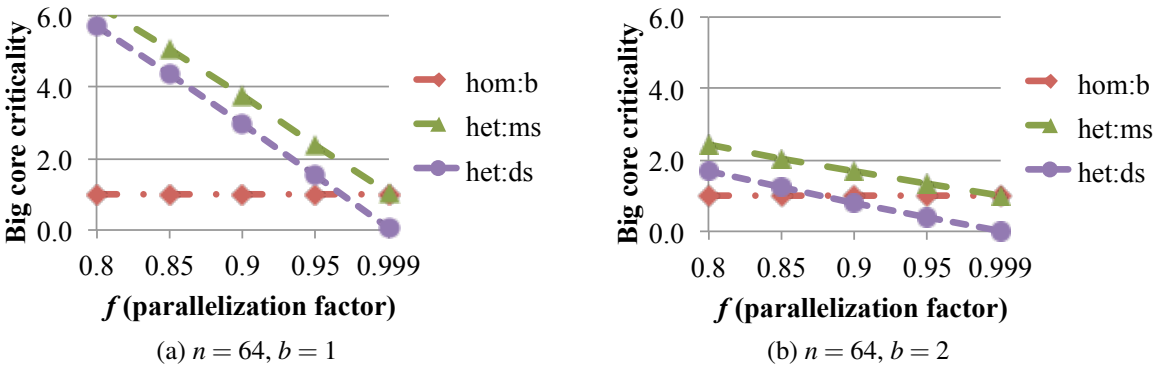
(a) $n = 64$, $b = 1$

(b) $n = 64$, $b = 2$

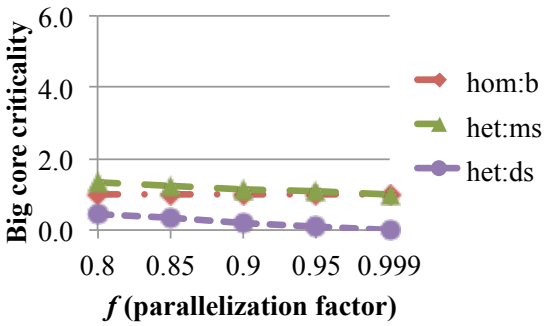
(c) $n = 64$, $b = 4$

Figure 31: Relative lifetime (MTTF) of various multicore models for $n = 64$ with parallelization fraction scaled between $f = 0.8$ and 0.999, and varying number of big cores (b) in the heterogeneous processors.

heterogeneous processors is changed by $b = 1$, 2, and 4 in the sub-plots. The baseline MTTF (= 1.0) is when the homogeneous processor of small cores is operating at 100% parallel executions. The MTTF curves of the homogeneous processor with simple cores are located above the MTTF = 1.0 line because of the serial part $1 - f$ that exercises only one simple core. Activating one core also produces better thermal field, so the failure rate of a core in serial phases is lower than that during parallel executions. More importantly, any cores in the homogeneous processor can be selected to perform serial operations (i.e., load sharing effect in the long term), and thus the serial phase is insignificant from the reliability perspective. The similar phenomena happen in the composed processor. For the same operating conditions, the homogeneous processor composed of big cores exhibits worse lifetime reliability since it spends relatively longer time on parallel phases that activate all cores in the processor.



(a) $n = 64$, $b = 1$ (b) $n = 64$, $b = 2$



(c) $n = 64$, $b = 4$

Figure 32: Relative reliability criticality of big cores in the heterogeneous multicores for $n = 64$ with parallelization factor between $f = 0.8$ and 0.999. The number of big cores (b) is varied in the sub-plots.

When the heterogeneous processor includes only one complex core, the MTTF decreases with smaller f (i.e., more serial operations) as shown in Figure 31(a) in contrast to the reliability behaviors of homogeneous processors. The reason can be explained with Figure 32 that plots the failure rate ratio between a big and small core per unit area. Since the only complex core in the heterogeneous processor has to handle all the sequential executions, increasing serial fraction of $1 - f$ puts more stresses on the complex core. The problem is especially worse with the maximum scheduling because the big core has to execute both serial and a part of parallel operations. In particular, the big core in the heterogeneous processor with maximum scheduling at $f = 0.8$ is $6.3 \times r$ times more likely to fail than a small core. For $r = 3$, it has about $19\times$ greater failure rate. Although a big core takes only a small portion of the total area when $n = 64$, uneven failure rate distribution between core types limits the overall lifetime of the heterogeneous processor.

If the heterogeneous processor includes multiple big cores, the reliability criticality of big cores is significantly reduced because of load sharing effect as shown in Figure 32(b). For instance, the heterogeneous processor with maximum scheduling shows 13% improvement in lifetime and 17% for dynamic scheduling at $f = 0.8$ by adding one more complex core, by comparing Figure 31(a) and (b). However, further increasing the number of big cores, $b = 4$ for example, rather diminishes the lifetime of heterogeneous processor with maximum scheduling especially for highly parallel workloads (e.g., $f = 0.999$) as shown in Figure 31(c). Increasing the big core count decreases peak parallel throughput, and it causes the processor to operate longer time in parallel phases. As the stresses shift from the big cores to simple cores by increasing b, including many big cores has a negative impact on lifetime reliability. The reliability of heterogeneous processor with dynamic scheduling continues to benefit from increasing the number of complex cores, but the similar effect happens when large number of complex cores are incorporated. Consequently, including a few number of big cores helps improve processor lifetime, but adding large number of big cores has an adverse impact on reliability as well as performance.

5.6 Application to Lifetime Reliability, Performance, and Energy Efficiency Tradeoffs

In this section, the performance, energy efficiency, and lifetime reliability models of multicore processors are applied to the simulation results of real benchmarks. For this analysis, Manifold cycle-level microarchitecture simulator [81] is used to collect performance counters of PARSEC and SPLASH-2 benchmarks [7], and McPAT [40] is used to estimate the power of exemplary complex and simple core models. Simulation outputs are extrapolated to construct hypothetical multicores, and the results are discussed. Table 12 summarizes the simulation setup used in this experiment.

Table 12: Simulation Setup for Performance and Power Estimation

Configurations	Description	
Core type	Complex core	Simple core
Issue width	6	1
Reorder buffer (ROB) size	128 entries	N/A
L1 cache size per core	32KB, 4-way assoc.	
L2 cache size per core	256KB, 8-way assoc.	
Clock frequency	2.0GHz	
Memory bandwidth	25GB/s	

5.6.1 Realistic Performance and Energy Scaling Models

A more realistic performance model from the work of Esmaeilzadeh et al. [17, 20] is considered in this section, instead of maximum performance speed-up assumed in Section 5.1. Eq. (32) shows that the performance scaling of a multicore processor is bounded by core throughput or memory bandwidth. In this equation, N is throughput-equivalent core count in unit of simple cores, and CPI_{active} is instruction latency during active periods. η represents core utilization factor to keep the core pipeline busy without stalls. BW_{mem} is the maximum memory bandwidth, and γ_{mem} is the rate of memory instructions with cache miss rates $\prod m$ that require off-chip memory accesses of data width d_{mem}.

$$Perf = min\left(N\frac{freq}{CPI_{active}}\eta, \frac{BW_{mem}}{\gamma_{mem}\times\prod m\times d_{mem}}\right) \tag{32}$$

Figure 33(a) plots the estimated performance speed-up of individual benchmarks when assuming that Amdahl's parallelization fraction is $f = 1.0$ (i.e., perfectly parallelizable); the performance speed-up is normalized to the throughput of each benchmark at $n = 1$. Figure 33(b) shows the actual Amdahl's scaling factor f of individual benchmarks. From the results in Figure 33, benchmarks can be categorized into the following classes:

- *Class I*: Blackscholes and Fluidanimate are highly scalable (i.e., good performance speed-up with increasing number of cores) but poorly parallelizable applications (i.e., relatively small parallelization fraction f).

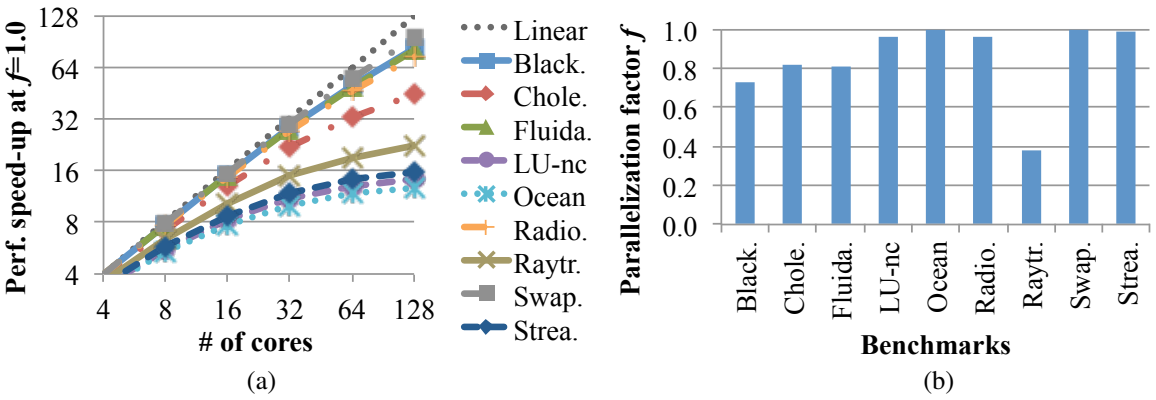


Figure 33: Benchmark characterization: (a) performance speed-up with increasing number of cores when assuming 100% parallel executions and (b) actual parallelization fraction f of individual benchmarks obtained from microarchitecture simulations.

- *Class II*: Ocean-nc and Streamcluter benchmarks are less scalable (i.e., saturating performance with increasing number of cores) but highly parallelizable (i.e., $f \approx 1.0$).

- *Class III*: Swaptions benchmark shows great multicore scalability and is also highly parallelizable.

- *Class IV*: Raytrace benchmark represents poorly scalable (i.e., limited performance speed-up with increasing core count) and highly serial applications (i.e., small parallelization fraction f).

- *Class V*: Other benchmarks show intermediate features.

From the simulation results, the area ratio between exemplary complex and small cores is estimated around $r = 3.2$. On average, the obtained performance ratio between two core types is around $s = 0.96 \times \sqrt{r}$ compared to Pollack's Rule [49], and the power ratio is estimated around $p = 0.94 \times \sqrt{r}^{\alpha}$ with respect to Chung's model [13]. When calculating energy efficiency, the dynamic power of cores is adjusted with utilization factor η in Eq. (32). Leakage power is repeatedly calculated based on the compact thermal estimation using Eq. (20) until estimated temperature and leakage power converge.

5.6.2 Application to Performance, Energy, and Lifetime Reliability Models of Multicore Processors

Figure 34(a) shows the performance speed-up of various multicore processors with different applications. The processor size is fixed at $n = 64$, and two big cores ($b = 2$) are included in the heterogeneous designs. Amdahl's scaling factor f is variant across benchmarks. For each benchmark, the results of multicore processors are normalized to that of the composed processor option.

In most cases, the heterogeneous and composed processors (1.0 line in the graph) outperform homogeneous implementations, producing higher throughput for both sequential and parallel executions except for the Class III applications. The Class III workloads are highly scalable and parallelizable, so they favor the homogeneous processor composed of small cores. Incorporating big cores in the heterogeneous processors reduces peak parallel throughput and thus diminishes the overall performance, compared to the homogeneous or composed processor made of simple cores. In contrast, the performance of the Class IV benchmarks is dominated by sequential executions. The Class III and IV show exactly opposite performance behaviors with the homogeneous processors. For the Class I type of applications (i.e., highly scalable but less parallelized), the overall performance speed-up is more governed by single-thread executions. The homogeneous processor comprised of small cores in this case shows inferior performance to other multicore configurations. The Class II benchmarks have large Amdahl's scaling factor ($f \approx 1.0$) but low multicore scalability. The homogeneous processor of big cores shows 13-17% lower performance than the other multicore processors, but the difference is limited in that the applications do not scale well with large number of cores.

The energy efficiency of multicore processors with different classes of benchmarks is plotted in Figure 34(b). The heterogeneous and composed processors show superior energy efficiency to the homogeneous configurations except for the Class III-type applications. The heterogeneous processor with maximum scheduling in overall has 2-8% lower

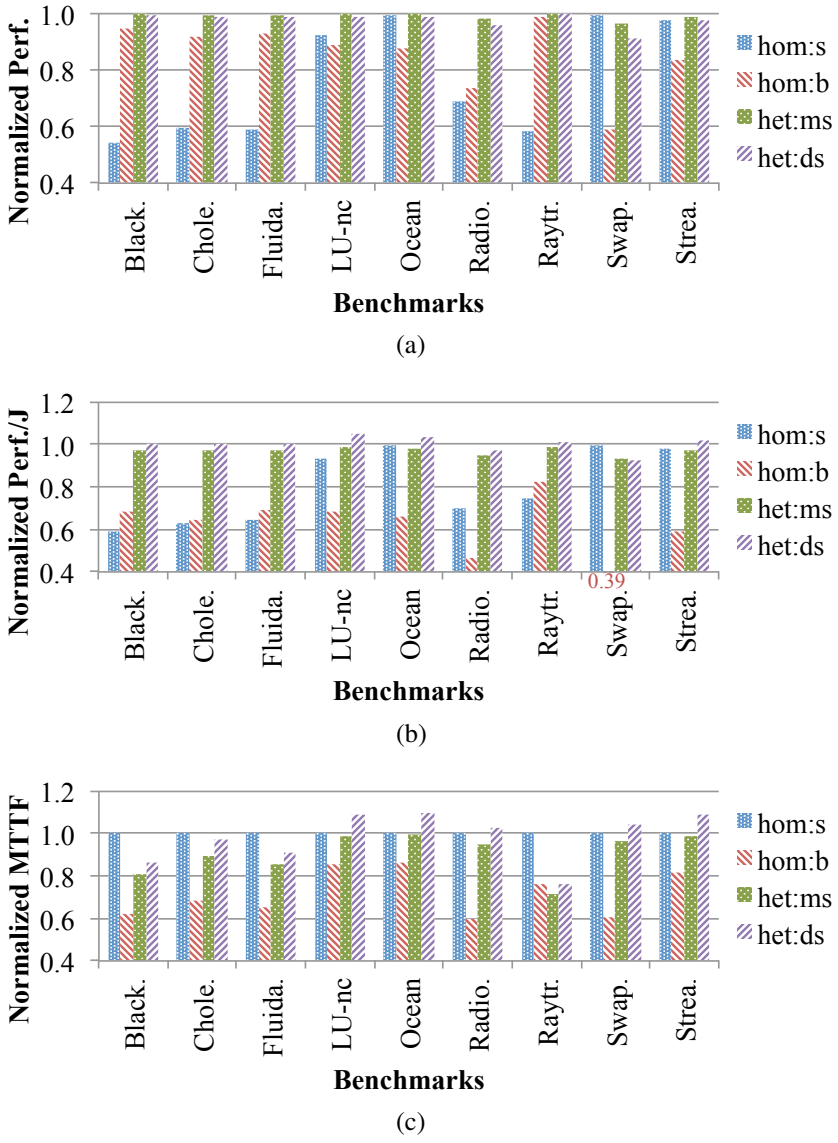


Figure 34: (a) Normalized performance, (b) energy efficiency, and (c) lifetime reliability of multicore processors for $n = 64$, $b = 2$, and $f = \textit{varying}$. The result of each benchmark is normalized to the composed processor option.

energy efficiency than the composed processor. The difference is caused by big cores that yield lower efficiency. The dynamic scheduling shows higher energy efficiency than the maximum scheduling. It particularly performs well with the Class II workloads (i.e., saturating performance with increasing number of cores), since turning off complex cores during parallel executions has minor impact on performance when $n = 64$. The homogeneous processor of complex cores has especially low energy efficiency, but it can be better

than the other homogeneous option if workloads consist of large fragment of serial executions such as the Class I and IV benchmarks. In sum, the heterogeneous processors produce similar performance speed-up and energy efficiency across different classes of applications, compared to the composed processor that represents an ideal implementation.

Figure 34(c) shows the normalized lifetime reliability of various multicore options. The homogeneous processor made of complex cores exhibits inferior lifetime reliability to other multicores in that it spends relatively longer time on executing parallel threads that require activating all cores in the processor. Notably, the heterogeneous processors have a serious reliability drawback when workloads are dominated by serial operations (i.e., Class I and IV). For these type of applications, stresses are excessively biased to the big cores. When the heterogeneous processor with maximum scheduling includes only one complex core instead of two, there is about 10% decrease in processor lifetime for the Class I benchmarks. The decrease becomes greater if applications are more bounded by serial executions. Thus, the results demonstrate that the overall lifetime reliability of heterogeneous processors can be limited by complex cores.

5.7 Summary

Microarchitectural heterogeneity has drawn attentions to enhance performance and energy efficiency. In this research, theoretical models are presented and discussed to understand the lifetime reliability consequences of heterogeneous multicores based on Amdahl's Law by extending prior work for performance and energy (and power) modeling Importantly, the performance, energy efficiency, and lifetime reliability of heterogeneous processors are traded as a function of processor size (n in unit of small cores), big core count (b), and Amdahl's scaling factor (f). The following summarizes key insights obtained from the analysis:

- When b/n ratio is small (e.g., one complex core and many small cores), this puts biased stresses on the complex core and thus is unfavorable for the lifetime reliability

especially when $f \ll 1.0$.

- Increasing processor size n (but fixed b) or decreasing Amdahl's scaling factor f shifts stresses from small cores to big cores in the heterogeneous processor and causes the biased stress (or reliability bottleneck) problem.

- When n is sufficiently large, adding a few big cores to the heterogeneous processor increases b/n, but the change is limited and therefore has a minor impact on performance and energy efficiency. In this case, increased b significantly reduces the reliability criticality of big cores and improves processor lifetime.

- However, further increasing the b/n ratio (i.e., more area dedicated to big cores) reduces peak parallel throughput, and extended execution time diminishes energy efficiency especially when $f \to 1.0$.

- For highly parallelizable workloads ($f \approx 1.0$), minimizing b/n ratio (e.g., only one complex core in the heterogeneous processor) is preferred.

CHAPTER VI

CONCLUSION

This dissertation addresses an important challenge of understanding how the processor physics including energy, power, thermal, and reliability is manifested in microarchitecture operations and application performance. Traditionally these problems were tackled separately at individual layers of system abstraction stack as depicted in Figure 35. However, developing future processors will require more *holistic system-level approaches* in addition to seeking solutions at each design layer of the processor system.

For example, leveraging Turbo Boosting [55] or Turbo Core [10] techniques is not solely about drawing more application performance. It incurs complicated physical problems such as power delivery, maximum junction temperature, thermal coupling, cooling, and device aging. It also has a significant impact on system-level metrics such as energy (or power) efficiency. All these outcomes are highly dependent on application characteristics and use cases. Therefore, it becomes increasingly important to view all these physical challenges and their coupled interactions with microarchitecture at the system level to characterize and develop future processors.

In this dissertation, three research topics are presented and discussed:

- Modeling and simulation method of interacting multicore processor physics
- Characterization and management of performance and lifetime reliability tradeoff
- Extending Amdahl's Law for understanding lifetime reliability, performance, and energy efficiency of heterogeneous processors

The first research topic proposes a novel simulation framework that orchestrates interactions between multiple physical models and microarchitecture simulators to enable

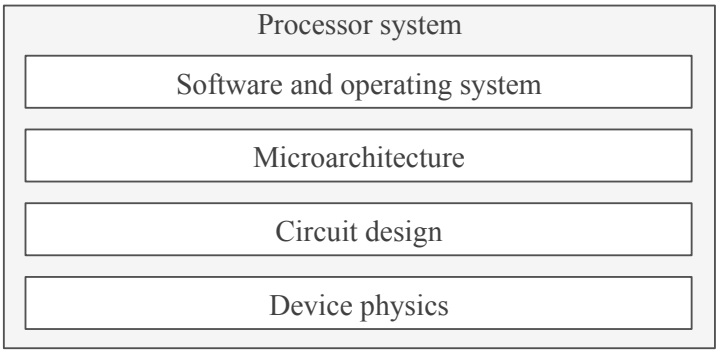


Figure 35: Design abstraction stack of a processor system from device to software layers. Developing and understanding future processors will require holistic inter-layer approaches in addition to seeking solutions at the individual design layers of the processor system.

research explorations at the intersection of application, microarchitecture, performance, energy, power, thermal, cooling, and reliability. This framework has been applied to a range of research problems requiring holistic system-level modeling and analysis.

The second research topic presents the characterization and management of performance-reliability tradeoff in multicore processors. This study connects device physics to application behaviors via microarchitectural adaptations, demonstrating inter-layer system characterizations and solutions. In this research, it is pointed out that reliability management cannot simply work to improve processor lifetime, but it must balance the tradeoff between performance and reliability.

The last study explores three important angles of multicore microarchitectures; performance, energy efficiency, and reliability. This research shows a necessity for the multi-dimensional analysis of microarchitectures since these metrics are traded as a function of various design parameters. In particular, this research characterizes heterogeneous multicore reliability by using Amdahl's Law.

In conclusion, the analysis of future microarchitectures must be holistic including power, energy, thermal, and reliability concerns since their coupled interactions are becoming major determinants of processor operations and performance. This dissertation fills the gap between microarchitecture and processor physics that have been treated as separate studies.

REFERENCES

[1] "Failure mechanisms and models for semiconductor devices." JEDEC Solid State Technology Association, JEDEC Publication JEP122C, Mar. 2006.

[2] ALMOOSA, N., SONG, W., WARDI, Y., and YALAMANCHILI, S., "A power capping controller for multicore processors," in *Proc. Am. Control Conf.*, pp. 4709–4714, Jun. 2012.

[3] ALMOOSA, N., SONG, W., YALAMANCHILI, S., and WARDI, Y., "Throughput regulation in multicore processors via IPA," in *Proc. IEEE Annu. Conf. Decis. Control*, pp. 7267–7272, Dec. 2012.

[4] AMDAHL, G., "Validity of the single processor approach to achieving large scale computing capability," in *Proc. AFIPS Spring Joint Comput. Conf.*, pp. 483–485, Apr. 1967.

[5] BARTOLINI, A., CACCIARI, M., TILLI, A., BENINI, L., and GRIES, M., "A virtual platform environment for exploring power, thermal, and reliability management control strategies in high-performance multicores," in *Proc. Great Lakes Symp. Very Large Scale Integr.*, pp. 311–316, May 2010.

[6] BEU, J., ROSIER, M., and CONTE, T., "Manager-Client Pairing: A framework for implementing coherence hierarchies," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, pp. 226–236, Dec. 2011.

[7] BIENIA, C., KUMAR, S., and LI, K., "PARSEC vs SPLASH-2: Quantitative comparison of two multithreaded benchmark suites on processors," in *Proc. IEEE Int. Symp. Workload Charact.*, pp. 47–56, Sep. 2008.

[8] BIENIA, C., KUMAR, S., SINGH, J., and LI, K., "The PARSEC Benchmark Suite: Characterization and architectural implications," in *Proc. Int. Conf. Parallel Archit. Compil. Tech.*, pp. 72–81, Oct. 2008.

[9] BINKERT, N., BECKMANN, B., BLACK, G., REINHARDT, S., SAIDI, A., BASU, A., HESTNESS, J., HOWER, D., KRISHNA, T., SARDASHTI, S., SEN, R., SEWELL, K., SHOAIB, M., VAISH, N., HILL, M., and WOOD, D., "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, May 2011.

[10] BRANOVER, A., FOLEY, D., and STEINMAN, M., "AMD Fusion APU: Llano," *IEEE Micro*, vol. 32, pp. 28–37, Jan. 2012.

[11] BROOKS, D., TIWARI, V., and MARTONOSI, M., "Wattch: A framework for architecture-level power analysis and optimizations," in *Proc. Annu. Int. Symp. Comput. Archit.*, pp. 83–94, May 2000.

[12] CAO, T., BLACKBURN, S., GAO, T., and MCKINLEY, K., "The yin and yang of power and performance for asymmetric hardware and managed software," in *Proc. Annu. Int. Symp. Comput. Archit.*, pp. 225–236, Jun. 2012.

[13] CHUNG, E., MILDER, P., HOE, J., and MAI, K., "Single-chip heterogeneous computing: does the future include custom logic, FPGAs, and GPGPUs?," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, pp. 225–236, Dec. 2010.

[14] COSKUN, A., ROSING, T., LEBLEBICI, Y., and MICHELI, G., "A simulation methodology for reliability analysis in multi-core SoCs," in *Proc. Great Lakes Symp. Very Large Scale Integr.*, pp. 95–99, May 2006.

[15] COSKUN, A., ROSING, T., MIHIC, K., MICHELI, G., and LEBLEBICI, Y., "Analysis and optimization of MPSoC reliability," *J. Low Power Electron.*, vol. 2, pp. 56–69, Jan. 2006.

[16] COSKUN, A., STRONG, R., TULLSEN, D., and ROSING, T., "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," in *Proc. Int. Conf. Meas. Model. Comput. Syst.*, pp. 169–180, Jun. 2009.

[17] ESMAEILZADEH, H., BLEM, E., AMANT, R., SANKARALINGAM, K., and BURGER, D., "Dark silicon and the end of multicore scaling," in *Proc. Annu. Int. Symp. Comput. Archit.*, pp. 365–376, Jun. 2011.

[18] FENG, S., GUPTA, S., ANSARI, A., and MAHLKE, S., "Maestro: Orchestrating lifetime reliability in chip multiprocessors," in *Proc. Conf. High Perform. Embedded Archit. Compil.*, pp. 186–200, Jan. 2010.

[19] GU, Z., ZHU, C., SHANG, L., and DICK, R., "Application-specific MPSoC reliability optimization," *IEEE Trans. Very Large Scale Integr.*, vol. 16, pp. 603–608, May 2008.

[20] GUZ, Z., BOLOTIN, E., KEIDAR, I., KOLODNY, A., MENDELSON, A., and WEISER, U., "Many-core vs many-thread machines: stay away from the valley," *Comput. Archit. Lett.*, vol. 8, pp. 25–28, Jan. 2009.

[21] HAN, Y., KOREN, I., and KRISHNA, C., "TILTS: A fast architecture-level transient thermal simulation method," *J. Low Power Electron.*, vol. 3, pp. 13–21, Apr. 2007.

[22] HARING, R., OHMACHT, M., FOX, T., GSCHWIND, M., SATTERFIELD, D., SUGAVANAM, K., COTEUS, P., HEIDELBERGER, P., BLUMRICH, M., WISNIEWSKI, R., GARA, A., CHIU, G., BOYLE, P., CHRIST, N., and KIM, C., "The IBM Blue Gene/Q compute chip," *IEEE Micro*, vol. 32, pp. 48–60, Dec. 2011.

[23] HILL, M. and MARTY, M., "Amdahl's law in the multicore era," *Comput.*, vol. 41, pp. 33–38, Jul. 2008.

[24] HSIEH, M., "A scalable simulation framework for evaluating thermal management techniques and the lifetime reliability of multithreaded multicore systems," in *Int. Green Comput. Conf. Workshops*, pp. 1–6, Jul. 2011.

[25] HSIEH, M., RIESEN, R., THOMPSON, K., SONG, W., and RODRIGUES, A., "A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration," *ACM SIGMETRICS Spec. Issue Int. Workshop Perform. Model. Benchm. Simul. High Perform. Comput. Syst.*, vol. 38, pp. 63–68, Mar. 2011.

[26] HSIEH, M., RIESEN, R., THOMPSON, K., SONG, W., and RODRIGUES, A., "SST: A scalable parallel framework for architecture-level performance, power, area, and thermal simulation," *Comput. J.*, pp. 181–191, Jul. 2011.

[27] HUANG, L. and XU, Q., "Characterizing the lifetime reliability of manycore processors with core-level redundancy," in *Proc. Int. Conf. Comput.-Aided Des.*, pp. 680–685, Nov. 2010.

[28] HUANG, L., YUAN, F., and XU, Q., "Lifetime reliability-aware task allocation and scheduling for MPSoC platforms," in *Proc. Conf. Des. Autom. Test in Europe*, pp. 51–56, Apr. 2009.

[29] HUANG, W., GHOSH, S., VELUSAMY, S., SANKARANARAYANAN, K., SKADRON, K., and STAN, M., "HotSpot: A compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. Very Large Scale Integr.*, vol. 14, pp. 501–513, May 2006.

[30] JOAO, J., SULEMAN, M., MUTLU, O., and PATT, Y., "Bottleneck identification and scheduling in multithreaded applications," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, pp. 223–234, Mar. 2012.

[31] JOAO, J., SULEMAN, M., MUTLU, O., and PATT, Y., "Utility-based acceleration of multithreaded applications on asymmetric CMPs," in *Proc. Annu. Int. Symp. Comput. Archit.*, pp. 154–165, Jun. 2013.

[32] KAHNG, A., LI, B., PEH, L., and SAMADI, K., "Orion 2.0: A fast and accurate NoC power and area model for early-state design space exploration," in *Proc. Conf. Des. Autom. Test in Europe*, pp. 423–428, Apr. 2009.

[33] KALLA, R., SINHAROY, B., STARKE, W., and FLOYD, M. *IEEE Micro*, pp. 7–15, Apr. 2010.

[34] KARL, E., BLAAUW, D., SYLVESTER, D., and MUDGE, T., "Reliability modeling and management in dynamic microprocessor-based systems," in *Proc. Annu. Des. Autom. Conf.*, pp. 1057–1060, Jul. 2006.

[35] KERSEY, C., RODRIGUES, A., and YALAMANCHILI, S., "A universal parallel frontend for execution driven microarchitecture simulation," in *Proc. Workshop Rapid Simul. Perform. Eval.*, pp. 25–32, Jan. 2012.

[36] KIM, H., LEE, J., LAKSHMINARAYANA, N., SIM, J., LIM, J., and PHO, T., "MacSim: A CPU-GPU heterogeneous simulation framework." HPArch Research Group, Georgia Institute of Technology.

[37] KIM, H., VITKOVSKIY, A., GRATZ, P., and SOTERIOU, V., "Use It Or Lose It: Wear-out and lifetime in future chip multiprocessors," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, pp. 136–147, Dec. 2013.

[38] KOUFATY, D., REDDY, D., and HAHN, S., "Bias scheduling in heterogeneous multi-core architectures," in *Proc. European Conf. Comput. Syst.*, pp. 125–138, Apr. 2010.

[39] KUMAR, R., TULLSEN, D., JOUPPI, N., and RANGANATHAN, P., "Heterogeneous chip multiprocessors," *IEEE Micro*, vol. 38, pp. 32–38, Nov. 2005.

[40] LI, S., AHN, J., STRONG, R., BROCKMAN, J., TULLSEN, D., and JOUPPI, N., "McPAT: Integrated power, area, timing modeling framework for multicore architectures," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, pp. 469–480, Dec. 2009.

[41] LIM, J., LAKSHMINARAYANA, N., KIM, H., SONG, W., YALAMANCHILI, S., , and SUNG, W., "Power modeling for GPU architectures using McPAT," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, pp. 26:1–26:24, Jun. 2014.

[42] LO, D. and KOZYRAKIS, C., "Dynamic management of TurboMode in modern multi-core chips," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, pp. 603–613, Feb. 2014.

[43] LOH, G., "Zesto: Rich and heavy microarchitecture simulation." *http://zesto.cc.gatech.edu*, Apr. 2009.

[44] LOH, G., SUBRAMANIAM, S., and XIE, Y., "Zesto: A cycle-level simulator for highly detailed microarchitecture exploration," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 53–64, Apr. 2009.

[45] LU, Z., LACH, J., STAN, M., and SKADRON, K., "Improved thermal management with reliability banking," *IEEE Micro*, vol. 25, pp. 40–49, Dec. 2005.

[46] MERCATI, P., BARTOLINI, A., PATERNA, F., ROSING, T., and BENINI, L., "Workload and user experience-aware dynamic reliability management in multicore processors," in *Proc. Annu. Des. Autom. Conf.*, pp. 1–6, Jun. 2013.

[47] MORAD, T., WEISER, U., KOLODNY, A., VALERO, M., and AYGUADE, E., "Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors," *Comput. Archit. Lett.*, vol. 5, pp. 14–17, Jun. 2006.

[48] MUTLU, O., STARK, J., WILKERSON, C., and PATT, Y., "Runahead Execution: An anlternative to very large instruction window for out-of-order processors," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, pp. 129–140, Feb. 2003.

[49] POLLACK, F., "New microarchitecture challenges in the coming generations of CMOS processor technologies," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, 1999.

[50] PRIYADARSHI, S., DAVIS, W. R., STEER, M., and FRANZON, P., "Thermal pathfinding for 3D ICs," *IEEE Trans. Compon. Packag. Manuf. Technol.*, vol. 4, pp. 1159–1168, May 2014.

[51] RAO, K., YALAMANCHILI, W. S. S., and WARDI, Y., "Temperature regulation in multicore processors using adaptive-gain integral controllers," in *IEEE Conf. Control Appl.*, Nov. 2015.

[52] REDDI, V., KANEV, S., KIM, W., CAMPANONI, S., SMITH, M., WEI, G., and BROOKS, D., "Voltage Smoothing: characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, pp. 77–88, Dec. 2010.

[53] RODRIGUES, A., HEMMERT, K., BARRETT, B., KERSEY, C., OLDFIELD, R., WESTON, M., RIESEN, R., COOK, J., ROSENFIELD, P., COOPER-BALIS, E., and JACOB, B., "The structural simulation toolkit," in *Proc. Int. Workshop Perform. Model. Benchm. Simul. High Perform. Comput. Syst.*, pp. 37–42, Mar. 2011.

[54] ROSENFELD, P., COOPER-BALIS, E., and JACOB, B., "DRAMsim2: A cycle accurate memory system simulator," *Comput. Archit. Lett.*, vol. 10, pp. 16–19, Jan. 2011.

[55] ROTEM, E., NAVEH, A., RAJWAN, D., ANANTHAKRISHNAN, A., and WEISSMANN, E., "Power-management architecture of the Intel microarchitecture code named Sandy Bridge," *IEEE Micro*, vol. 32, pp. 20–27, Feb. 2012.

[56] SAJJADI-KIA, H. and ABABEI, C., "A new reliability evaluation methodology with application to lifetime-oriented circuit design," *IEEE Trans. Device Mater. Reliab.*, vol. 13, pp. 192–202, Mar. 2013.

[57] SEKAR, D., NAEEMI, A., SARVARI, R., DAVIS, J., and MEINDL, J., "IntSim: A CAD tool for optimization of multi-level interconnect network," in *Proc. Int. Conf. Comput.-Aided Des.*, pp. 560–567, Nov. 2007.

[58] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Simulation infrastructure for power and temperature modeling in many-core processors and linear system modeling," in *Proc. SRC TECHCON*, Sep. 2011.

[59] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Architecture simulation framework for 3D ICs," in *Proc. SRC TECHCON*, Sep. 2012.

[60] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Reliability implications of power and thermal-constrained operation in asymmetric multicore processors," in *Proc. Dark Silicon Workshop*, Jun. 2012.

[61] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Lifetime reliability and accelerated execution," in *Proc. SRC TECHCON*, Sep. 2013.

[62] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Architectural Reliability: Lifetime reliability characterization and management of many-core processors," *Comput. Archit. Lett.*, Jul. 2014.

[63] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Energy Introspector: A parallel, composable framework for integrated power-reliability-thermal modeling for multicore architectures," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 143–144, Mar. 2014.

[64] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Energy Introspector: Standard physical library interface for full-system microarchitecture and multi-physics simulations," in *Proc. Workshop Model. Simul. Exascale Syst. Appl.*, Aug. 2014.

[65] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Lifetime reliability characterization and management of many-core processors," in *Proc. SRC TECHCON*, Sep. 2014.

[66] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "KitFox: multi-physics libraries for integrated power, thermal, and reliability simulations of multicore microarchitecture," *IEEE Trans. Compon. Packag. Manuf. Technol.*, Oct. 2015.

[67] SONG, W., MUKHOPADHYAY, S., and YALAMANCHILI, S., "Managing performance-reliability tradeoffs in multicore processors," in *Proc. IEEE Int. Reliab. Phys. Symp.*, pp. 3C.1.1–3C.1.7, Apr. 2015.

[68] SONG, W., MUKHOPADHYAY, S., YALAMANCHILI, S., and RODRIGUES, A., "Instruction-based energy estimation methodology for asymmetric manicure processor simulations," in *Proc. Int. Conf. Simul. Tools and Techn.*, Mar. 2012.

[69] SONG, W., MUKHOPADHYAY, S., YALAMANCHILI, S., and RODRIGUES, A., "KitFox: Multi-physics libraries for integrated power, thermal, and reliability simulation of multicore microarchitecture." *http://manifold.gatech.edu/projects/kitfox*, Apr. 2015.

[70] SRIDHAR, A., VINCENZI, A., RUGGIERO, M., BRUNSCHWILER, T., and ATIENZA, D., "3D-ICE: Fast compact transient thermal modeling for 3d ics with inter-tier liquid cooling," in *Proc. Int. Conf. Comput.-Aided Des.*, pp. 463–470, Nov. 2010.

[71] SRINIVASAN, J., ADVE, S., BOSE, P., and RIVERS, J., "The case for lifetime reliability-aware microprocessors," in *Proc. Annu. Int. Symp. Comput. Archit.*, pp. 276–287, Jun. 2004.

[72] SRINIVASAN, J., ADVE, S., BOSE, P., and RIVERS, J., "The impact of technology scaling on lifetime reliability," in *Proc. Int. Conf. Dependable Syst. Netw.*, pp. 177–186, Jun. 2004.

[73] SRINIVASAN, J., ADVE, S., BOSE, P., and RIVERS, J., "Exploiting structural duplication for lifetime reliability enhancement," in *Proc. Int. Symp. Comput. Archit.*, pp. 520–531, Jun. 2005.

[74] SRINIVASAN, J., ADVE, S., BOSE, P., and RIVERS, J., "Lifetime Reliability: Toward an architectural solution," *IEEE Micro*, vol. 25, pp. 70–80, May 2005.

[75] SULEMAN, M., MUTLU, O., QURESHI, M., and PATT, Y., "Accelerating critical section execution with asymmetric multi-core architectures," in *Proc. Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, pp. 253–264, Mar. 2009.

[76] SUN, C., CHEN, C., KURIAN, G., WEI, L., MILLER, J., AGARWAL, A., PEH, L., and STOJANOVIC, V., "DSENT: A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling," in *Proc. IEEE/ACM 6th Int. Symp. Netw.-on-Chip*, pp. 201–210, May 2012.

[77] THOZIYOOR, S., AHN, J., MONCHIERO, M., BROCKMAN, J., and JOUPPI, N., "Comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Proc. Annu. Int. Symp. Comput. Archit.*, pp. 51–62, Jun. 2008.

[78] TOMUSK, E., O'BOYLE, M., DUBACH, C., and GOODACRE, J., "Practicalities of design space exploration with gem5 and McPAT," in *Conf. High Perform. and Embedded Archit. and Compil. Comput. Syst. Week*, Oct. 2012.

[79] TOTONI, E., BEHZAD, B., GHIKE, S., and TORRELLAS, J., "Comparing the power and performance of Intel's SCC to state-of-the-art CPUs and GPUs," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 78–87, Mar. 2012.

[80] WAN, Z., KIM, Y., and JOSHI, Y., "Compact modeling of 3D stacked die inter-tier microfluidic cooling under non-uniform heat flux," in *Proc. Int. Mech. Eng. Congr. Exhib.*, pp. 911–917, Sep. 2012.

[81] WANG, J., BEU, J., BHEDA, R., CONTE, T., DONG, Z., KERSEY, C., RASQUINHA, M., RILEY, G., SONG, W., XIAO, H., XU, P., and YALAMANCHILI, S., "Manifold: A parallel simulation framework for multicore systems," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, pp. 106–115, Mar. 2014.

[82] WHITE, M. and BERNSTEIN, J., "Microelectronics Reliability: Physics-of-failure based modeling and lifetime evaluation." JPL Publication 08-5 2/08, NASA Jet Propulsion Laboratory, 2008.

[83] WOO, D. and LEE, H., "Extending Amdahl's law for energy-efficient computing in the many-core era," *Comput.*, vol. 41, pp. 24–31, Dec. 2008.

[84] WOO, S., OHARA, M., and TORRIE, E., "The SPLASH-2 Programs: Characterization and methodological considerations," in *Proc. Annu. Int. Symp. Comput. Archit.*, pp. 24–36, Jun. 1995.

[85] XIANG, Y., CHANTEM, T., DICK, R., HU, X., and SHANG, L., "System-level reliability modeling for MPSoCs," in *Proc. IEEE/ACM/IFIP Hardw./Softw. Codes. Syst. Synth.*, pp. 297–306, Oct. 2010.

[86] XIAO, H., WAN, Z., YALAMANCHILI, S., and JOSHI, Y., "Leakage power characterization and minimization in 3D stacked multi-core chips with microfluidic cooling," in *Proc. Annu. IEEE Semicond. Therm. Meas. Manage. Symp.*, pp. 207–212, Mar. 2014.

[87] YAMAMOTO, A. and ABABEI, C., "Unified reliability estimation and management of NoC-based chip multiprocessors," *Microprocess. Microsyst.*, vol. 38, pp. 53–63, Nov. 2013.

[88] YU, J., ZHOU, W., YANG, Y., ZHANG, X., and YU, Z., "Many-core processors granularity evaluation by considering performance, yield, and lifetime reliability," *IEEE Trans. Very Large Scale Integr.*, Oct. 2014.

[89] ZANINI, F., ATIENZA, D., COSKUN, A., and MICHELI, G., "Optimal multiprocessor SoC thermal simulation via adaptive differential equation solvers," in *Proc. IFIP Int. Conf. Very Large Scale Integr.*, pp. 139–146, Oct. 2009.

[90] ZYUBAN, V., TAYLOR, S., CHRISTENSEN, B., HALL, A., GONZALEZ, C., FRIEDRICH, J., CLOUGHERTY, F., TETZLOFF, J., and RAO, R., "IBM POWER7+ design for higher frequency at fixed power," *IBM J. Res. Dev.*, pp. 1:1–1:18, Dec. 2013.